BIBLIOTECAS DE AXIOMÁTICAS - CONCEITOS E RESULTADOS PARA SISTEMAS ALGÉBRICOS

João Jorge da Costa Vieira Ramires

Orientador: Professor Doutor João Jorge Ribeiro Soares Gonçalves de Araújo
Co-Orientador: Professor Doutor David Martins de Matos

Tese especialmente elaborada para obtenção do Grau de
Doutor em Álgebra Computacional

Outubro de 2022

Júri de avaliação

Presidente

- Doutora Maria João Chaves Marques da Cunha Oliveira, Professora Associada com Agregação, do Departamento de Ciências e Tecnologia, da Universidade Aberta.

Vogais

- Doutor Manuel Baptista Branco, Professor Associado do Departamento de Matemática da Universidade de Évora;

- Doutor Rui Castanheira de Paiva, Professor Adjunto do Departamento de Matemática do Instituto Politécnico do Leiria;

- Doutora Gilda Maria Saraiva Dias Ferreira, Professora Auxiliar do Departamento de Ciências e Tecnologia da Universidade Aberta;

- Doutor Wolfram Florian Bentz, Professor Auxiliar do Departamento de Ciências e Tecnologia da Universidade Aberta;

- Doutor João Jorge Ribeiro Soares Gonçalves de Araújo, Professor Catedrático do Departamento de Matemática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (orientador).

Agradecimentos

Aos meus orientadores, o Professor Doutor João Jorge Ribeiro Soares Gonçalves de Araújo (Universidade Nova de Lisboa), e ao Professor Doutor David Martins de Matos (Instituto Superior Técnico), pelo continuo apoio à minha investigação.

Aos colegas que contribuíram no desenvolvimento deste projeto, nomeadamente a Jorge Raminhos, Yves Robert e Matheus Lobo e aos estudantes Afonso Simplício e Inês Jesus (Universidade Nova de Lisboa) e Pedro Almeida (Universidade Aberta).

Aos seguintes investigadores, que direta ou indiretamente, contribuíram de forma decisiva para este projeto: Professor Michael Kinyon (University of Denver), Professor Peter Jipsen (Chapman University) e Professor Peter Höfner (Australian National University).

João Ramires
Universidade Aberta, Portugal
Outubro de 2022

Resumo

Em 1996, EQP, um programa de computador, resolveu o Problema de Robbins, um problema colocado nos anos 30 e que tinha derrotado alguns dos maiores algebristas do século XX. Em julho de 2022, Enigma, uma ferramenta de inteligência artificial aplicada à demonstração automática de teoremas produzida por Josef Urban (Czech Technical University) no âmbito de uma bolsas do Conselho Europeu de Investigação (ERC) que lhe foi concedida para o efeito, demonstrou autonomamente 75% dos teoremas na base de dados Mizar, um repositório contendo grande parte dos teoremas mais comuns, nomeadamente os teoremas lecionados numa licenciatura ou mestrado de matemática pura.

Os tempos são de grande mudança e a questão natural é esta: como podemos colocar estas ferramentas a ajudar o progresso da matemática?

A resposta que existe hoje consiste em fazer uma pergunta ao computador e ter uma resposta (como aconteceu com o EQP e com a Enigma).

Mas será possível desenhar um programa para autonomamente descobrir nova matemática? Aqui há um problema. Já hoje temos computadores capazes de descobrir milhões e milhões de novos teoremas (isto é, sentenças válidas numa determinada teoria) de forma autónoma; mas qual o valor dessa matemática e que impacto tem ou poderá ter? Como garantir que esses teoremas gerados não são triviais, desinteressantes ou inúteis? E o que fazer com teoremas que não se compreendem? Que impacto pode ter imenso ruído ou imensos ininteligíveis?

Têm sido avançadas diferentes propostas de resolver a questão natural colocada acima. Algumas delas já revelaram falhar. Outras ainda estão a seguir o seu caminho. O objetivo desta tese é dar uma resposta completamente diferente à pergunta e, como prova de conceito, gerar muitos teoremas, novos, inteligíveis e interessantes.

De forma simples (a realidade é bastante mais complexa que a descrição que se segue), a resposta que avançamos nesta tese é a seguinte. Vamos construir um plano em que num dos eixos temos teorias e no outro eixo temos teoremas (ie, sentenças válidas sobre determinados objetos em determinada teoria). No início o nosso plano tem apenas alguns pontos (A, B), significando que na teoria A o resultado B é verdadeiro. Por exemplo: Teoria de Grupos, *É comutativo o grupo em que cada elemento é inverso de si próprio.* Agora, o programa, autonomamente verifica se (X, B) é verdadeiro (onde B é um teorema conhecido fixo e X está a percorrer todas as teorias do nosso repositório). Sempre que encontra uma prova para (Q, B), temos um teorema novo, que é totalmente compreensível e familiar, e que (no caso de Q generalizar A) será tão ou mais interessante que o teorema (A, B) original. Por exemplo, ao correr o nosso programa no repositório, imediatamente surge o teorema: Teoria de Semigrupos Inversos, *É comutativo o semigrupo inverso em que cada elemento é inverso de si próprio.*

Acontece que os grupos são uma pequena classe dentro dos semigrupos inversos, uma estrutura que aparece naturalmente na geometria, e o teorema de grupos referido é dado em qualquer curso de teoria de grupos, enquanto o teorema de semigrupos inversos aparentemente nem sequer aparece na literatura.

De igual modo, dado o teorema (A, B), podemos procurar teoremas (A, X), em que X está a percorrer teoremas conhecidos do repositório. No caso de se provar (A, P), teremos um teorema famoso P, até então conhecido para a teoria T, e que agora se verifica ocorrer na teoria A também. Uma vez mais o resultado é familiar, inteligível e novo.

Um utilizador que descobre um teorema T numa axiomática A, pode colocar T no nosso sistema que irá procurar pares (X, T), quando X percorre a base de dados de axiomáticas do nosso repositório. Tendo como resposta (A_1, T), (A_2, T),..., (A_n, T), o utilizador ficará assim a saber que o seu teorema é um fenómeno muito mais geral e portanto pode procurar uma teoria geral Γ que contenha todas as teorias A_i como casos particulares e na qual o seu teorema T ainda seja verdadeiro. Mais interessante ainda, esta busca da teoria Γ pode ser feita autonomamente pelo computador.

De forma análoga, um matemático pode ser levado à introdução de uma nova classe A. O nosso sistema vai procurar pares (A, X), onde X são teoremas na base de dados, pelo que poucos minutos depois de ter identificado a nova classe, o utilizador poderá já ter vários teoremas familiares, possivelmente não triviais e uma base muito mais sólida para prosseguir a sua investigação. Quando antes a introdução de uma nova classe significava o início de um longo e penoso caminho a provar novos teoremas, a início elementares, e depois crescentemente complexos, o nosso sistema permite obter um corpo significativo de teoremas, podendo alguns ter provas altamente complexas, não obstante soando familiares e compreensíveis.

É esta a resposta que esta tese propõe para a pergunta mais natural do momento: *como se pode domar, tornar significativa e útil, a capacidade ilimitada de gerar novos teoremas dos computadores?*

Palavras chave: Modelo, Teorema, Demonstração automática de teoremas, Prover9, MarcieX

Abstract

In 1996, EQP, a computer program, solved the Robbins Problem, a problem posed in the 1930s and that had defeated some of the top algebristas of the 20th century. In July 2022, Enigma, an artificial intelligence tool applied to the automatic demonstration of theorems produced by Josef Urban (Czech Technical University) as part of a European Research Council (ERC) scholarship granted to him to this end, demonstrated autonomously 75% of the theorems in the Mizar database, a repository containing much of the most common theorems, namely the theorems taught in a bachelors degree or masters degree in pure mathematics.

The times are of great change and the natural question is this: how can we put these tools to help the progress of mathematics?

The answer that exists today is to ask a question to the computer and have an answer (as happened with EQP and Enigma).

But it will be possible to design a program to autonomously discover new mathematics? Theres a problem here. Today we have already computers capable of discovering millions and millions of new theorems (i.e. sentences valid in a theory) autonomously; but whats the value of this mathematics and what impact will it have? How to ensure that these generated theorems are not trivial, uninteresting or useless? And what to do with theorems we do not understand? What impact can have a lot of trivial or deeply unintelligible statements?

Different proposals have been put forward to address this natural issue. Some of them already failed. Others are still following their way. The purpose of this thesis is to give a completely different answer to this main problem and, as proof of concept, generate many theorems that are new, intelligible and interesting.

The answer we have advanced in this thesis is as follows. Lets build a plane where in one of the axes we have theories and on the other axis we have theorems (i.e., valid sentences on certain objects in a given theory). At the beginning our plane has only a few points (A, B), meaning that in theory A result B is true. For example: A is Theory of Groups, and B is *The group in which each element is inverse of itself is commutative.* Now the program autonomously checks whether (X, B) is true (where B is a fixed known theorem and X is going through all the theories of our repository). Whenever you find a proof for (Q, B), we have a new theorem, which is totally understandable and familiar, and that (in the case of Q generalizing A or Q not related to A) will be as much or more interesting than the original theorem (A, B). For example, when running our program in the repository, it immediately appears the theorem: Q is Theory of Inverse Semigroups, *It is commutative the inverse semigroup where each element is inverse of itself.*

It turns out that groups are a small class within the inverse semigroups, a structure that appears naturally in geometry, and the group theory theorem referred to above is taught in any course of

group theory, while the inverse semigroup theorem apparently does not even appear in the literature.

Similarly, we can look for theorems (A, X), where X is going through our repository of theorems. In the case we get a proof for (A, P), we will have a famous P theorem, until then known for the Theory T (the starting point was (T, P)), and that now happens to hold for theory A as well. Once again, the result is familiar, intelligible and new.

The first approach will give all theories in which a given result B is true; the second approach will give all results that hold in a given theory A.

If a mathematician discovers a theorem P in an axiomatic T, then P can be added to our system that will look for pairs (X, P), when X spans our library of mathematical theories. With the answer $(A_1, P), (A_2, P), ..., (A_n, P)$, the user will thus know that his or her theorem is a much more general phenomenon and therefore can look for a general theory Γ that contains all A_i theories as particular cases and in which theorem P still holds. Even more interestingly, this search for theory Γ can be done independently by the computer.

Analogously, a mathematician can be led to the introduction of a new class A. Our system will look for pairs (A, X), where X are theorems in the database, so a few minutes after identifying the new class, the user may already have several familiar theorems, and possibly a much stronger basis for further research. When, in the old days, the introduction of a new class meant the beginning of a long and often painful path to prove new elementary theorems at first, and then increasingly complex, our system allows us to obtain upfront a significant body of theorems, some of which may have highly complex proofs, but all of them will sound familiar and understandable.

This is the answer that this thesis proposes to the most natural question of the moment: *how can you control, tame, make meaningful and useful, the unlimited capacity to generate new computer theorems?*

Keywords: Model, Theorem, Automated theorem proving, Prover9, MarcieX

Conteúdo

1 Introdução **1**

1.1 Lógica matemática 4

1.2 Demonstração automática de teoremas 6

2 Estado de arte **11**

2.1 Bases de dados de axiomáticas 11

2.2 Ferramentas computacionais 12

2.2.1 Vampire 12

2.2.2 Waldmeister 13

2.2.3 Prover9 14

3 Solução proposta **17**

3.1 Extração de tabelas 17

3.1.1 Representante da classe de isomorfismo 18

3.1.2 Imersão de modelos 18

3.1.3 Busca de modelos 19

3.2 Extração de axiomáticas 19

3.2.1 A base de dados de classes de álgebras 20

3.2.2 A representação interna das axiomáticas 21

3.2.3 O conjunto parcialmente ordenado de teorias 24

3.3 Base de dados de teoremas 25

3.4 Arquitetura 25

4 A geração de novos teoremas **27**

4.1 Teoremas geradores 27

4.2 Redução do número de axiomas 29

4.3 Localização de uma identidade 34

4.4 Grafos 40

4.5 Inclusão de classes 42

5 MarcieX **231**

5.1 Models and Theories 231
5.2 Original theorems 238
5.3 Generated theorems 239
5.4 Original / Generated theorems 240
5.5 Download de aplicação 241
5.6 Model Explorer 242

6 Teoremas **249**

6.1 Teoremas geradores 249
6.1.1 Semigroups 249
6.1.2 Inverse semigroup 254
6.1.3 Conjugacy 274
6.1.4 Diassociative automorphic loop 275
6.1.5 F-quasigroup 276
6.1.6 Quasigroup 276
6.1.7 Demonic Refinement Algebra 277
6.1.8 Left Kleene Algebra 280
6.1.9 Modal I-Semiring 281
6.1.10 Relation Algebra 303
6.1.11 Cancellation semigroup with a power-like map 321
6.1.12 Group with a power-like map 322
6.1.13 Hoop 322
6.1.14 Relational Lattice 327
6.1.15 Hoare Triples 329
6.2 Teoremas gerados 332
6.2.1 Medial quasigroups 332
6.2.2 Left Bol loop 333
6.2.3 Right Bol loop 334
6.2.4 Left conjugacy closed loop 334
6.2.5 Right conjugacy closed loop 334
6.2.6 RIF loop 335
6.2.7 ARIF loop 335
6.2.8 Extra loop 336
6.2.9 BCI-algebras 337

6.2.10 BCK-algebras 354
6.2.11 Commutative BCK-algebras 371
6.2.12 Pocrims 393
6.2.13 Directoids 394
6.2.14 Normal bands 415
6.2.15 Order algebras 434
6.2.16 Semigroups with zero 455
6.2.17 Orthoimplication algebra 473
6.2.18 Orthomodular implication algebra 492
6.2.19 Quasi-implication algebra 513
6.2.20 Left closure semigroup 531
6.2.21 Left closure monoid 546
6.2.22 Inverse semigroup 563
6.2.23 Domain range semigroup 589
6.2.24 Antidomain monoid 594
6.2.25 Semilattice pseudo-complemented semigroup 617
6.2.26 Closable semilattice pseudo-complemented semigroup 637
6.2.27 Cancellative semigroup 659
6.2.28 Involutory quandle 679
6.2.29 Steiner quasigroup 693
6.2.30 Steiner loop 711
6.2.31 Double Ward quasigroup 733
6.2.32 Associative dialgebra 747
6.2.33 Associative dimonoid 750
6.2.34 Bounded lattice 752
6.2.35 Commutative dimonoid 760
6.2.36 Complemented lattices 772
6.2.37 Complemented modular lattices 782
6.2.38 Diassociative semigroup 791
6.2.39 Dimonoid 792
6.2.40 Disemigroup 795
6.2.41 Doppelsemigroup 800
6.2.42 Duplicial semigroup 801
6.2.43 Involutive lattices 803
6.2.44 Left disemigroup 805
6.2.45 Multiplicative semilattices 808

6.2.46 Near-ring . . . 809
6.2.47 Near-semiring . . . 810
6.2.48 Neardistributive lattices . . . 810
6.2.49 Ockham algebras . . . 813
6.2.50 Ortholattices . . . 829
6.2.51 Quandles . . . 831
6.2.52 Right disemigroup . . . 843
6.2.53 Shells . . . 845
6.2.54 Skew lattices . . . 848
6.2.55 Strong doppelsemigroup . . . 852
6.2.56 Commutative regular rings . . . 855
6.2.57 Division ring . . . 862
6.2.58 Fields . . . 868
6.2.59 Integral domains . . . 875
6.2.60 Near-fields . . . 881
6.2.61 Near-ring . . . 882
6.2.62 Near-rings with identity . . . 891
6.2.63 Skew-fields . . . 896
6.2.64 Domain semigroup . . . 902
6.2.65 Domain monoid . . . 905
6.2.66 Digroup . . . 908
6.2.67 Bilattice . . . 913
6.2.68 Boolean monoid . . . 914
6.2.69 Commutative lattice-ordered monoids . . . 916
6.2.70 Commutative lattice-ordered rings . . . 916
6.2.71 Commutative lattice-ordered semigroups . . . 918
6.2.72 De Morgan monoids . . . 919
6.2.73 Distributive lattice ordered semigroups . . . 920
6.2.74 F-rings . . . 922
6.2.75 Lattice-ordered rings . . . 924
6.2.76 Lattice-ordered semigroups . . . 926
6.2.77 Multiplicative lattices . . . 927
6.2.78 Normal valued lattice-ordered groups . . . 928
6.2.79 Sequential algebras . . . 929
6.2.80 Commutative integral ordered monoid . . . 931
6.2.81 Commutative ordered monoids . . . 933

6.2.82 Commutative ordered semigroups 935
6.2.83 Commutative partially ordered monoids 936
6.2.84 Commutative partially ordered semigroups 938
6.2.85 Commutative regular rings 939
6.2.86 Commutative residuated partially ordered monoids 939
6.2.87 Integral ordered monoids 941
6.2.88 Ordered monoids with zero 943
6.2.89 Ordered semigroup 945
6.2.90 Ordered semilattice 946
6.2.91 Partially ordered groups 948
6.2.92 Partially ordered monoids 951
6.2.93 Partially ordered semigroups 953
6.2.94 Residuated partially ordered monoids 954
6.2.95 Totally ordered monoids 956

7 Conclusões e trabalho futuro **959**

8 Anexos **961**
8.1 Base de dados de modelos 961
8.2 Base de dados de teorias 969
8.3 Package CREAM 1019

Capítulo 1

Introdução

Atribui-se ao matemático Irving Kaplansky a conjetura de que as técnicas de raciocínio automático, permitirão aos matemáticos do século XXI resolver problemas tão profundos e complexos que qualquer matemático do século XX os acharia ininteligíveis. Note-se que um matemático do final do século XIX poderia não acompanhar os detalhes, mas não se sentiria perdido numa conferência atual de teoria de grupos, álgebra em geral, geometria, etc. O ponto de Kaplansky é que isso deixará de ocorrer na passagem para a matemática assistida por raciocínio automático do século XXI.

Em 1996, o New York Times reportou que no Argonne National Laboratory, um programa de computador denominado EQP e desenhado por Bill McCune em colaboração com Larry Wos (um antigo aluno de Kaplansky), conseguiu resolver um problema de matemática. Tratava-se do Problema de Robbins, colocado nos anos 30 e que havia derrotado alguns dos maiores matemáticos do século XX como A. Tarski e D. Pigozzi. Este sucesso foi o culminar de um enorme esforço que tinha começado muitos anos antes com o sonho de Leibnitz que seria a criação de uma linguagem não ambígua onde as questões pudessem ser formuladas, e depois seria criada uma máquina capaz de as interpretar e decidir. A procura dessa linguagem ocupou os matemáticos durante 200 anos terminando na Lógica de Primeira Ordem (LPO). Já a procura da máquina capaz de decidir sentenças em LPO exigiu o trabalho coordenado de imensos investigadores, incluindo muitos Prémios Turing, mas a última pedra, a ferramenta que de facto permitiu pela primeira vez decidir uma questão formulada em LPO e cuja resposta os matemáticos não conheciam, ficou reservada ao extraordinário grupo de investigadores reunidos no Argonne National Laboratory especificamente para essa tarefa.

Simon Colton (University of Edinburgh) como parte da sua tese de doutoramento em 2000, escreveu um programa capaz de gerar conceitos e formar conjeturas sobre eles que depois eram provados ou refutados por ferramentas de raciocínio automático. Esta ferramenta tinha por objetivo substituir os matemáticos quer na conjetura, quer na demonstração de teoremas. O programa denominado HR (de Hardy-Ramanujan) fez um ensaio com a teoria de grupos: trabalhou durante dez minutos, gerou inúmeros conceitos, 40 mil conjeturas, conseguindo provar ou refutar todas elas, à exceção de 200; estas foram depois passadas a especialistas em teoria de grupos que não as conseguiram decidir. Esta tese causou um grande impacto, foi distinguida como a melhor do ano e mais tarde como a melhor da década, e foi publicada na serie da *Springer Distinguished Dissertations*.

Parecia que a visão de Kaplansky estava a ponto de se concretizar e, pior ainda, que os matemáticos (ou, pelo menos, os algebristas) estariam no desemprego, do mesmo modo que a invenção das calculadoras desempregou as pessoas que antes calculavam tabelas de logaritmos ou funções

trigonométricas. Mas de facto não foi bem assim. Rapidamente, se percebeu que HR não tinha capacidade de gerar conceitos ou conjeturas com a profundidade necessária a equiparar-se ao trabalho dos especialistas em teoria de grupos.

Em 2018, a revista Science publicou o artigo relativo ao AlphaZero, um programa de computador que sem acesso ao conhecimento acumulado nas estratégias de muitos jogos, conseguia rapidamente treinar-se de forma a derrotar os campeões nesses jogos. Isto sugeriu a David McAllester (Toyota Technological Institute at Chicago) a possibilidade de um MathZero, um programa que, conhecendo apenas alguns fundamentos de matemática fosse capaz de gerar matemática a um nível que batesse qualquer matemático humano. No entanto, parece que o MathZero tem um problema de fundo. O Xadrez, por exemplo, é um jogo com regras fixas e um objetivo claro; tudo o que o computador faz é compreensível por qualquer pessoa (o mesmo, claro, não se pode dizer da razão *por que o faz* e por isso há grandes mestres a fazer engenharia inversa das jogadas do AlphaZero para tentar perceber o que o leva a ganhar tanto e assim perceber como podem os humanos jogar melhor). Mas a matemática não é assim. Em matemática há total liberdade para criar teorias, introduzir objetos novos nas teorias existentes, conjeturar, inventar uma nova notação, provar teoremas, e o valor do resultado (ao contrário do Xadrez em que é objetivo) é em muitos casos subjetivo (o que leva as pessoas a valorizar mais o Teorema X do que o Y? Ou a valorizar mais a investigação na área A do que na Z?). Além disso, um teorema é uma sentença válida numa determinada axiomática, mas que é inteligível por alguém. Já hoje os computadores conseguem gerar sentenças verdadeiras em determinada teoria, mas qual o ponto ou utilidade se ela não se perceber?

Os pontos seguintes tentam esquematizar um pouco as diferenças entre o novo conhecimento gerado no Xadrez e na Matemática:

- Objetos. Xadrez: Fixos (peças); Matemática: Ilimitados;
- Objetivo. Xadrez: Ganhar; Matemática: Muitos, de muitas naturezas e vagos ou indefinidos (o que é matemática interessante?);
- Transições. Xadrez: Número finito de movimentos bem definidos executados por um conjunto finito de peças; Matemática: Infinitos caminhos de profundidade ilimitada;
- Compreensão do resultado. Xadrez: Total; Matemática: Um teorema sobre uma teoria que ninguém conhece, envolvendo objetos que ninguém percebe, possivelmente em notação desconhecida, e uma prova tão grande (milhões de páginas) que ninguém conseguirá acompanhar, dificilmente acrescentará algo à matemática;
- Inovação. Xadrez: Os humanos podem aprender novas jogadas, novas estratégias, novas aberturas, etc.; Matemática: Uma tal panóplia de resultados e provas, umas geniais, outras fastidiosas e não informativas, outras ininteligíveis, num mundo cujos limites estarão sempre a mudar (a geração de novos teoremas será mais rápida que a capacidade para os tentar digerir).

Mesmo que a visão de um MathZero se concretizasse hoje, seria difícil vir a ter utilidade prática para a investigação em matemática. Para uma sentença ser um teorema, não basta que seja válida nas hipóteses: é necessário que seja inteligível por alguém. Uma máquina capaz de gerar matemática ininteligível, vai gerar muitas sentenças verdadeiras, mas parece difícil que gere verdadeiro conhecimento matemático.

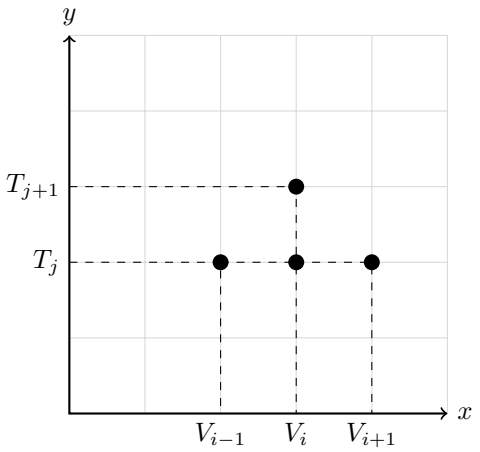
Evidentemente, um MathZero com capacidades sobrehumanas de demonstração terá imensa utilidade como oráculo a responder a perguntas: P=NP? A hipótese de Riemann é verdadeira? A conjetura de Goldbach? Mas isso é só (embora não seja pouco) uma diferença de grau relativamente às ferramentas que existem hoje e que já respondem a perguntas. O programa EQP respondeu à pergunta que nenhum matemático conseguiu responder.

O problema base desta tese era dar um passo além dos programas que respondem a perguntas. A

questão é saber *como podemos ter um programa a gerar autonomamente matemática inteligível e portanto útil*?

A resposta de Simon Colton foi construir um programa que gerava conceitos e conjeturas autonomamente. A resposta desta tese é diferente. Aqui usamos os conceitos e teoremas existentes para guiar a procura de novos teoremas. Dado um teorema sobre um conceito que envolve operações binárias, o teorema pode ser testado em muitas outras teorias que usem pelo menos uma operação binária.

Esquematicamente, esta tese produz um plano em que no eixo dos YY temos teorias diferentes e no eixo dos XX temos teoremas válidos em alguma axiomática. Se o ponto (T_j, V_i) está marcado significa que o teorema T_j é verdadeiro na teoria V_i. O que o programa faz automaticamente é completar a tabela: ver se o teorema T_j é também válido nas teorias V_{i-1}, V_{i+1}, etc.



Isto significa que cada novo teorema é *familiar*, envolve objetos conhecidos em alguma parte da matemática, envolve teoremas cujo enunciado é conhecido ou até clássico. E se o resultado for considerado interessante pelos matemáticos, podem explorar a prova para ver se tem informação nova e relevante. Além disso, o facto do mesmo teorema ser válido em várias teorias diferentes é um convite a procurar uma teoria geral que contenha todas as anteriores como caso particular e na qual seja verdadeiro o teorema em causa. E mesmo essa busca pela generalização comum pode ser automatizada.

Esta tese parte de um corpo de teorias e teoremas, e sobre ele é montado um sistema que poderá ser livremente usado pelos matemáticos em alguma das seguintes formas:

- Introdução de um teorema concreto (novo ou conhecido) e pedir generalizações a outras classes de objetos existentes na base de dados de teorias;
- Introdução de uma nova teoria (possivelmente nova) e pedir ao sistema que prove teoremas análogos a teoremas da base de dados;
- Introdução de uma nova teoria e identificação das classes da base de dados que estão contidas ou contêm a teoria dada;
- Introdução de modelos e identificação na literatura de artigos onde existam cópias isomorfas a esse modelo;
- Para os utilizadores avançados, capazes de definir computacionalmente tipos de teoremas (por exemplo: A classe de álgebras C pode ser definida por uma única identidade; Um objeto pertence à classe de álgebras A se e só se pertence à classe B e não contém o modelo M) podem fazer o

varrimento de todas as classes de álgebras desta base de dados para encontrar teoremas do tipo desejado;
- Base para a produção de bibliotecas de álgebras. Esta faceta já foi útil nas teses de vários alunos deste doutoramento (Fernando Ferreira [17], Choiwah Chow [18], Carlos Sousa [19], Rui Barradas Pereira [19]).

Como prova de conceito, foram definidos tipos de teoremas diferentes, sendo posteriormente percorrida toda a base de dados de forma a gerar teoremas novos, embora reconhecíveis como familiares pelos matemáticos.

Este projeto teve várias fases:

1. Construção de uma base de dados de modelos para permitir rapidamente encontrar na literatura qualquer modelo introduzido pelo utilizador;

2. Construção da base de dados de teorias e teoremas;

3. Geração de novos teoremas;

4. Construção de um interface para o utilizador.

Ao contrário do HR que gerou muitos teoremas de grupos que não suscitaram grande interesse aos especialistas, este sistema gerou teoremas novos, análogos a teoremas existentes (seja teoremas cujo enunciado generaliza *ipsis verbis* a outras classes, seja gerando tipos de teoremas).

Neste capítulo vão ser abordados alguns fundamentos de lógica e de demonstração automática de teoremas. No capítulo 2 será descrito o estado de arte de algumas bases de dados de axiomáticas e ferramentas computacionais. O capítulo 3 descreverá a solução proposta e no capítulo 4 serão apresentadas algumas estratégias para a geração de novos teoremas. O capítulo 5 descreve o interface MarcieX e no capítulo 6 são mostrados alguns teoremas usados na geração de novos teoremas e respetivos teoremas gerados. No capítulo 7 serão discutidas conclusões e trabalho futuro. Os resultados obtidos podem ser consultados em `www.marciex.com`.

1.1 Lógica matemática

Nesta secção vão ser descritos alguns fundamentos de Lógica Proposicional (LP) e de Lógica de Primeira Ordem (LPO) [1, 2, 3, 4].

A Lógica Proposicional permite representar proposições com um valor lógico verdadeiro ou falso. A construção de proposições formalmente corretas, normalmente conhecidas por fórmulas bem formadas (*fbf*), obedece a um conjunto de regras de sintaxe. Usando o formalismo de Backus-Naur (BNF) vem que

```
<fbf> ::= <formula_atomica> | <formula_composta>
<formula_atomica> ::= verdadeiro | falso | <variavel>
<formula_composta> ::= ¬<fbf> | (<fbf> <operador_logico> <fbf>)
<operador_logico> ::= ∧ | ∨ | → | ↔
<variavel>::= P | Q | R | ...
```

Atribuindo valores lógicos às variáveis proposicionais e às operações que envolvem os operadores lógicos, é obtida uma interpretação da *fbf*. Uma interpretação μ de uma *fbf* α que origina um valor lógico verdadeiro para α, é chamada de modelo de α e escreve-se $\mu \models \alpha$.

Uma *fbf* é chamada válida se e só se for verdadeira para todas as interpretações, e inválida se e

só se existir pelo menos uma interpretação para a qual é falsa. Uma *fbf* é chamada inconsistente se e só se for falsa para todas as interpretações, e consistente se e só se existir pelo menos uma interpretação para a qual é verdadeira. Uma fórmula válida é em particular consistente e uma fórmula inconsistente é em particular inválida.

Seja Σ um conjunto de *fbf* α_i, e μ um modelo de cada α_i, tem-se que μ é modelo de Σ. Diz-se que a *fbf* α é uma consequência lógica de Σ se e só se todos os modelos de Σ também forem modelos de α, e escreve-se $\Sigma \models \alpha$.

O conjunto de todas as consequências lógicas de Σ é conhecido por teoria de Σ e escreve-se $Teo(\Sigma) = \{\alpha \mid \Sigma \models \alpha\}$. Os elementos de Σ são chamados axiomas da teoria e aos elementos da teoria, teoremas de Σ. Tem-se ainda que a *fbf* α é consequência lógica de Σ se e só se $(\alpha_1 \wedge ... \wedge \alpha_i) \rightarrow \alpha$ for válida. A *fbf* α é consequência lógica de Σ se e só se $\alpha_1 \wedge ... \wedge \alpha_i \wedge \neg\alpha$ for inconsistente.

Se para duas *fbf* α e γ, a *fbf* $\alpha \leftrightarrow \gamma$ é válida, então diz-se que α e γ são logicamente equivalentes e escreve-se $\alpha \equiv \gamma$. Isto permite a substituição de uma *fbf* α por outra *fbf* γ, em que a substituição de uma *fbf* por outra logicamente equivalente não altera o valor de verdade sob nenhuma interpretação.

Através de regras de inferência é possível gerar uma conclusão (nova *fbf*) a partir de um conjunto de premissas (*fbf* conhecidas). Se existir uma sequência de *fbf* pertencente a Σ ou gerada a partir de regras de inferência que origine α, então à sequência dá-se o nome de prova de α. Se existe prova de α a partir de Σ, escreve-se $\Sigma \vdash \alpha$.

Se for garantido que as regras de inferência asseguram o valor lógico verdade para a conclusão a partir de premissas verdadeiras, diz-se que as regras são consistentes. Logo, se existe prova de α a partir de Σ, então α é uma consequência lógica de Σ (se $\Sigma \vdash \alpha$ então $\Sigma \models \alpha$). Por outro lado, se uma *fbf* for uma consequência lógica de um conjunto de *fbf*, então deve ser possível provar isso, e diz-se que as regras são completas. Logo, se α é uma consequência lógica de Σ, então pode-se provar α a partir de Σ (se $\Sigma \models \alpha$ então $\Sigma \vdash \alpha$). Assim, para um conjunto de regras de inferência consistente e completo, diz-se que tudo o que pode ser demonstrado é verdadeiro e tudo o que é verdadeiro pode ser demonstrado. Neste contexto, os métodos de raciocínio lógico baseados em modelos podem ser substituídos por métodos baseados na utilização de regras de inferência.

A Lógica de Primeira Ordem contém elementos novos que permitem estender a Lógica Proposicional. Em LP, as proposições mais simples (fórmulas atómicas) são consideradas como unidades indivisíveis, onde não são estudadas as propriedades lógicas nem as relações que possam existir dentro de cada parte que constitui uma proposição, em que cada parte não seja ela própria uma proposição. Em LPO as fórmulas atómicas são constituídas por unidades mais pequenas (termos) unidas por um símbolo que as relaciona (predicado).

Por exemplo, para a declaração $x \geq y$ não pode ser atribuído um valor de verdade visto que depende do valor das variáveis x, y e da relação $\geq$. Em LPO consideram-se unidades mais pequenas como os valores de x, y e o símbolo de relação $\geq$, obrigando ainda à definição de um conjunto de valores possíveis (domínio).

A construção de *fbf* para LPO obedece ao seguinte conjunto de regras de sintaxe, em BNF. Os símbolos predicado/função são representados respectivamente por P_i/f_i para aridade P^j/f^j.

```
<fbf> ::= <formula_atomica> | <formula_composta>
<formula_atomica> ::= <simbolo_predicado>(<lista_termos>)
<formula_composta> ::=  ¬<fbf> | (<fbf><operador_logico><fbf>) |
        <quantificador><variavel><fbf>
<simbolo_predicado> ::= P_1^0 | P_2^0 | P_3^0 | ... | P_1^1 | ... | P_1^2 | ...
<operador_logico> ::= ∧ | ∨ | → | ↔
<quantificador> ::= ∀ | ∃
<lista_termos> ::= λ | <termo> | <termo>,<lista_termos>
<termo> ::= <constante> | <variavel> | <termo_composto>
<termo_composto> ::= <simbolo_funcao>(<lista_termos>)
<simbolo_funcao> ::= f_1^0 | f_2^0 | f_3^0 | ... | f_1^1 | ... | f_1^2 | ...
<constante> ::= verdadeiro | falso | a | b | c | ...
<variavel> ::= x | y | z | ...
```

Uma interpretação da *fbf* é conseguida pela atribuição de valores a termos/predicados/funções, dentro de um determinado domínio, e atribuição de valores lógicos às operações que envolvem os operadores lógicos e os quantificadores[1]. Os conceitos referidos em LP de modelo, validade, inconsistência, consequência lógica e inferência também se aplicam à LPO.

Se α é uma consequência lógica de Σ, então pode-se provar α a partir de Σ (se $\Sigma \models \alpha$ então $\Sigma \vdash \alpha$) e diz-se que as regras são completas. Se a partir de Σ se pode provar α, então α é uma consequência lógica de Σ (se $\Sigma \vdash \alpha$ então $\Sigma \models \alpha$) e diz-se que as regras são consistentes.

1.2 Demonstração automática de teoremas

Têm sido desenvolvidos métodos que permitem a automatização de geração de provas, normalmente conhecidos por métodos de raciocínio automático. Nesta secção vão ser introduzidos alguns conceitos que servem de base à compreensão desses métodos, nomeadamente ao método de resolução.

A aplicação da resolução é possível apenas se as *fbf* estiverem na forma clausal, que corresponde a uma conjunção de cláusulas, sendo cada cláusula formada por um literal ou uma disjunção de literais, e em que cada literal corresponde a uma *fbf* atómica ou à sua negação. Normalmente no método de resolução é utilizada prova por absurdo, que consiste em adicionar às premissas a negação da conclusão para gerar uma contradição (cláusula vazia, $\square$), e diz-se que é uma prova por refutação (se $\Sigma \models \alpha$ então $\Sigma \cup \{\neg\alpha\} \vdash \square$).

A forma clausal representa um conjunto de disjunções de conjuntos de conjunções de cláusulas de uma *fbf*. Por exemplo, em LP a *fbf*

$(P \vee \neg P) \wedge (Q \vee R)$

é representada na forma clausal pelo conjunto $\{\{P, \neg P\}, \{Q, R\}\}$.

A *fbf* do exemplo anterior diz-se que se encontra na forma conjuntiva normal. No conjunto que a representa, tem-se que as disjunções foram substituídas por virgulas que separam os subconjuntos e as conjunções por virgulas que separam os literais dentro dos subconjuntos. O par $(P \vee \neg P)$ é ainda chamado de par de literais em conflito, e pode ser removida da *fbf* sem influenciar o valor de verdade.

Na resolução é utilizada uma prova por refutação, aplicada à *fbf* na forma clausal, para verificar a cláusula vazia. Por exemplo, em LP para provar

$\{\{P\}, \{\neg P, Q\}, \{\neg Q, R\}\} \vdash R$,

[1] Uma definição detalhada de interpretação pode ser consultada em [2].

por refutação tem-se que

$\{\{P\}, \{\neg P, Q\}, \{\neg Q, R\}, \{\neg R\}\} \vdash \square$.

Em cada etapa da prova tem-se sequencialmente o seguinte[2]:

$\{\neg R\} \cup \{\neg Q, R\}$ origina $\{\neg Q\}$,

$\{\neg Q\} \cup \{\neg P, Q\}$ origina $\{\neg P\}$ e

$\{\neg P\} \cup \{P\}$ origina $\square$.

Todas as *fbf* em LP podem ser representadas com uma formula logicamente equivalente em forma clausal. No entanto, em LPO as *fbf* com quantificadores necessitam de ser convertidas, onde os quantificadores passam a estar no princípio das formulas, seguido de um processo denominado por *skolemização*, onde as variáveis que estão quantificadas pelo quantificador existencial são removidas e substituídas por constantes e funções. Por exemplo, em LPO para o conjunto de cláusulas seguinte,

$\{\{p(x, f(y)), p(g(y), f(a)), q(c, z)\}$,

$\{\neg p(g(a), z), r(x, a)\}$,

$\{p(x, f(y))\}$,

$\{\neg p(g(a), z)\}\}$

é possível substituir x por $g(a)$ e z por $f(y)$ simultaneamente para a obtenção de literais complementares, obtendo-se assim

$\{\{p(g(a), f(y)), p(g(y), f(a)), q(c,\ f(y))\}$,

$\{\neg p(g(a),\ f(y)), r(g(a), a)\}$,

$\{p(g(a),\ f(y))\}$,

$\{\neg p(g(a), f(y))\}\}$,

o que origina

$\{p(g(y),\ f(a)), q(c, f(y)), r(g(a), a)\}$.

O processo de resolução em LPO pode ser resumido nas seguintes etapas:
1) acrescentar a negação da conclusão às premissas;
2) os quantificadores são movidos para o início das *fbf*, é aplicada a *skolemização* e é obtido um conjunto de cláusulas;
3) seleção de duas cláusulas, com substituições que permitam a obtenção de literais complementares, conhecido por *unificador mais geral*. A conjunção das cláusulas anteriores origina uma cláusula conhecida por *resolvente*;
4) continuação para a etapa 3 até ser obtida a cláusula vazia.

Um exemplo do processo de resolução é apresentado de seguida, para provar que a *relação de inclusão entre conjuntos é transitiva*.

Sejam os conjuntos A, B, C, com $A \subseteq B$, tal que $\forall x(x \in A \rightarrow x \in B)$ (premissa) e $A \subseteq B \wedge B \subseteq C \rightarrow A \subseteq C$ (conclusão). A premissa pode ser representado como $\forall x \forall y(S(x, y) \leftrightarrow (\forall z(M(z, x) \rightarrow M(z, y))))$ e a conclusão como $\forall x \forall y \forall z(S(x, y) \wedge S(y, z) \rightarrow S(x, z))$ com o símbolo de predicado S a significar *subconjunto de* e o símbolo de predicado M como *membro de*. É necessária a construção de um conjunto de cláusulas sobre o qual o processo de resolução vai ser aplicado. Inicialmente, a conclusão é negada e colocada juntamente com a premissa:

[2]Por aplicação da regra de inferência *modus ponens*. É conhecido que para as *fbf* $P \rightarrow Q, P \vdash Q$.

$\forall x \forall y(S(x,y) \leftrightarrow (\forall z(M(z,x) \rightarrow M(z,y)))) \wedge \neg(\forall x \forall y \forall z(S(x,y) \wedge S(y,z) \rightarrow S(x,z)))$.

As formulas são transformadas em várias etapas. O processo vai ser descrito para a premissa e posteriormente para a conclusão negada.

$\forall x \forall y(S(x,y) \leftrightarrow (\forall z(M(z,x) \rightarrow M(z,y))))$

Etapa 1. Substituição de implicações, com $I \leftrightarrow J \equiv (I \rightarrow J) \wedge (J \rightarrow I)$, $I \rightarrow J \equiv \neg I \vee J$ e $\neg(\neg I) \equiv I$.

1.1.a $\forall x \forall y(S(x,y) \rightarrow (\forall z(M(z,x) \rightarrow M(z,y))))$

1.1.b $\forall x \forall y((\forall z(M(z,x) \rightarrow M(z,y))) \rightarrow S(x,y))$

1.2.a $\forall x \forall y(\neg S(x,y) \vee (\forall z(\neg M(z,x) \vee M(z,y))))$

1.2.b $\forall x \forall y(\neg(\forall z(\neg M(z,x) \vee M(z,y))) \vee S(x,y))$

Etapa 2. Cada símbolo de negação é aplicado a cada *fbf* atómica, com o quantificador Q como $\neg Qx \equiv Q\neg x$.

2.1.a $\forall x \forall y(\neg S(x,y) \vee (\forall z(\neg M(z,x) \vee M(z,y))))$

2.1.b $\forall x \forall y((\exists z \neg(\neg M(z,x) \vee M(z,y))) \vee S(x,y))$

2.2.a $\forall x \forall y(\neg S(x,y) \vee (\forall z(\neg M(z,x) \vee M(z,y))))$.

2.2.b $\forall x \forall y((\exists z(M(z,x) \wedge \neg M(z,y))) \vee S(x,y))$.

Etapa 3. As variáveis são renomeadas de modo a que cada quantificador seja referente a uma única variável.

3.1.a $\forall x \forall y(\neg S(x,y) \vee (\forall z(\neg M(z,x) \vee M(z,y))))$

3.1.b $\forall x \forall y((\exists z(M(z,x) \wedge \neg M(z,y))) \vee S(x,y))$

Etapa 4. Os quantificadores existenciais são eliminados e substituídos por funções *Skolem*. Se as variáveis dos quantificadores existenciais não pertencem ao âmbito dos quantificadores universais, as variáveis são substituídas por um novo símbolo de constante e o quantificador é eliminado. Caso contrário, a variável do quantificador existencial é substituída por um símbolo de função definida com variáveis quantificadas universalmente, e o quantificador existencial é eliminado.

4.1.a $\forall x \forall y(\neg S(x,y) \vee (\forall z(\neg M(z,x) \vee M(z,y))))$

4.1.b $\forall x \forall y((M(f(x,y),x) \wedge \neg M(f(x,y),y)) \vee S(x,y))$

Etapa 5. Os quantificadores universais são movidos para o início das *fbf*.

5.1.a $\forall x \forall y \forall z(\neg S(x,y) \vee (\neg M(z,x) \vee M(z,y)))$

5.1.b $\forall x \forall y((M(f(x,y),x) \wedge \neg M(f(x,y),y)) \vee S(x,y))$

Etapa 6. As *fbf* são colocadas no formato normal de conjunções.

6.1.a $\forall x \forall y \forall z(\neg S(x,y) \vee \neg M(z,x) \vee M(z,y))$

6.1.b $\forall x \forall y((M(f(x,y),x) \vee S(x,y)) \wedge (\neg M(f(x,y),y) \vee S(x,y)))$

Etapa 7. Os termos são selecionados.

7.1.a $\neg S(x,y) \vee \neg M(z,x) \vee M(z,y)$

7.1.b $(M(f(x,y),x) \vee S(x,y))$

7.1.c $(\neg M(f(x,y),y) \vee S(x,y))$

Etapa 8. Seleção do conjunto de cláusulas.

8.1.a $\{\neg S(x,y), \neg M(z,x), M(z,y)\}$

8.1.b $\{M(f(x,y),x), S(x,y)\}$

8.1.c $\{\neg M(f(x,y),y), S(x,y)\}$

São mostradas de seguida, as etapas aplicadas à negação da conclusão.

$\neg(\forall x \forall y \forall z(S(x,y) \wedge S(y,z) \rightarrow S(x,z)))$

$\exists x \exists y \exists z \neg((S(x,y) \wedge S(y,z)) \rightarrow S(x,z))$

Etapa 1. Substituição de implicações.

$\exists x \exists y \exists z \neg(\neg(S(x,y) \wedge S(y,z)) \vee S(x,z))$

Etapa 2. Cada símbolo de negação é aplicado a *fbf* atómica.

$\exists x \exists y \exists z((S(x,y) \wedge S(y,z)) \wedge \neg S(x,z))$

Etapa 3. As variáveis são renomeadas de modo a que cada qualificador seja referente a uma única variável.

$\exists x \exists y \exists z(S(x,y) \wedge S(y,z) \wedge \neg S(x,z))$

Etapa 4. Os quantificadores existenciais são eliminados e substituídos por funções *Skolem*.

$S(c1,c2) \wedge S(c2,c3) \wedge \neg S(c1,c3)$

Etapa 8. Seleção do conjunto de cláusulas.

8.1.a $\{S(c1,c2)\}$

8.1.b $\{S(c2,c3)\}$

8.1.c $\{\neg S(c1,c3)\}$

O conjunto de cláusulas está construído e vai ser aplicado o processo de resolução. É alcançada uma prova a partir das cláusulas 1 a 6, dado que o conjunto de cláusulas derivado das premissas juntamente com o conjunto de cláusulas derivado da negação da conclusão é uma cláusula vazia na etapa 13.

1 $\{\neg S(x,y), \neg M(z,x), M(z,y)\}$

2 $\{M(f(x,y),x), S(x,y)\}$

3 $\{\neg M(f(x,y),y), S(x,y)\}$

4 $\{S(c1,c2)\}$

5 $\{S(c2,c3)\}$

6 $\{\neg S(c1,c3)\}$

Das cláusulas 4 $\{S(c1,c2)\}$ e 1 $\{\neg S(x,y), \neg M(z,x), M(z,y)\}$, x é substituído por $c1$, y por $c2$ para se obter o *unificador mais geral* (escreve-se $s = \{x/c1, y/c2\}$) tem-se que $\{S(c1,c2)\}$, $\{\neg S(c1, c2), \neg M(z, c1), M(z,c2)\}$ e é obtido o *resolvente*

7 $\{\neg M(x,c1), M(x,c2)\}$

Das cláusulas 5 $\{S(c2,c3)\}$ e 1 $\{\neg S(x,y), \neg M(z,x), M(z,y)\}$ com $s = \{x/c2,\ y/c3,\ z/x\}$ vem que $\{S(c2,c3)\}$, $\{\neg S(c2,c3), \neg M(x,c2),\ M(x,c3)\}$ e

8 $\{\neg M(x,c2), M(x,c3)\}$

Das cláusulas 6 $\{\neg S(c1,c3)\}$ e 2 $\{M(f(x,y),x), S(x,y)\}$ com $s = \{x/c1, y/c3\}$ vem que $\{\neg S(c1,c3)\}$, $\{M(f(c1,c3),c1), S(c1,c3)\}$ e

9 $\{M(f(c1,c3),c1)\}$

Das cláusulas 6 $\{\neg S(c1, c3)\}$ e 3 $\{\neg M(f(x, y), y), S(x, y)\}$ com $s = \{x/c1,\ y/c3\}$ vem que $\{\neg S(c1,\ c3)\}$, $\{\neg M(f(c1, c3), c3), S(c1, c3)\}$ e

10 $\{\neg M(f(c1, c3), c3)\}$

Das cláusulas 9 $\{M(f(c1, c3), c1)\}$ e 7 $\{\neg M(x, c1), M(x, c2)\}$ com $s = \{x/f(c1,\ c3)\}$ vem que $\{M(f(c1, c3), c1)\}$, $\{\neg M(f(c1, c3), c1),\ M(f(c1, c3),\ c2)\}$ e

11 $\{M(f(c1, c3), c2)\}$

Das cláusulas 8 $\{\neg M(x, c2), M(x, c3)\}$ e 10 $\{\neg M(f(c1, c3), c3)\}$ com $s = \{x/f(c1, c3)\}$ vem que $\{\neg M(f(c1, c3), c2), M(f(c1, c3), c3)\}$, $\{\neg M(f(c1, c3),\ c3)\}$ e

12 $\{\neg M(f(c1, c3), c2)\}$

Finalmente, das cláusulas 12 $\{\neg M(f(c1, c3),\ c2)\}$ e 11 $\{M(f(c1, c3),\ c2)\}$ vem

13 $\square$.

Capítulo 2

Estado de arte

2.1 Bases de dados de axiomáticas

Um repositório bem conhecido é o TPTP (*Thousands of Problems for Theorem Provers*)[1]. A partir de uma sintaxe comum, não só é possível testar demonstradores automáticos de teoremas quanto à sua velocidade e capacidade de resolver problemas relativamente a uma grande variedade de áreas, mas também testar vários sistemas simultaneamente[2].

Outro repositório de teoremas é o Mizar[3], uma base de dados de matemática formalizada, escrita numa linguagem própria que permite a verificação de cada um dos passos da demonstração. É um grande projeto cujo objetivo é formalizar (e verificar) toda a matemática, detetando no processo alguns teoremas que se pensavam estar certos, mas cujas provas têm falhas. Talvez o momento alto do projeto foi quando Kevin Buzzard (Imperial College) formalizou uma prova de Peter Scholze, Medalha Fields e muito interessado em garantir que um teorema fundamental para o seu novo projeto (Condensed Mathematics) estava certo. O processo de formalização demorou cerca de um ano, mas no final o computador aprovou a prova[4]. Note-se que formalizar um teorema pode ser uma dissertação de mestrado, uma tese de doutoramento, ou um projeto de investigação extremamente exigente.

Mas o que é importante sublinhar é que se a visão subjacente ao Mizar e a outros sistemas análogos (Lean, Isabelle, etc.) se concretizasse hoje, o conhecimento matemático estaria verificado, mas seria o mesmo que existe hoje (ou um pouco diferente apenas por no processo se encontrarem erros em teoremas que se pensava estarem certos). Mesmo o projeto Enigma[5] de Josef Urban conseguiu resultados surpreendentes, mas que consistiram em descobrir autonomamente provas para teoremas enunciados no Mizar, não para provar novos teoremas importantes.

Esta tese distingue-se de todos estes projetos por introduzir um sistema que procura e encontra teoremas novos, profundos e interessantes.

[1] TPTP - Thousands of Problems for Theorem Provers, a library of test problems for automated theorem proving systems. `http://www.tptp.org`

[2] System on TPTP, services for solving problems and recommending systems. `http://www.tptp.org/cgi-bin/SystemOnTPTP`

[3] Mizar, a mathematical concepts database. `http://mizar.org`

[4] `https://www.quantamagazine.org/lean-computer-program-confirms-peter-scholze-proof-20210728/`

[5] `https://people.ciirc.cvut.cz/~urbanjo3/`

2.2 Ferramentas computacionais

Nesta altura há dezenas de demonstradores automáticos (Prover9, E, Vampire, etc.) e ferramentas para construir modelos (*finite model builders*). Uns com mais sucesso que outros, muitos que deixaram de ser suportados, etc. Vamos apenas referir alguns.

2.2.1 Vampire

Vampire[6] é um demonstrador de teoremas para LPO, mas também pode construir modelos finitos, sendo ainda um dos demonstradores mais rápidos. O Vampire aceita a notação TPTP como entrada e utiliza dezenas de regras de inferência [15]. É apresentado de seguida um exemplo de um conjunto de axiomas e a respectiva notação TPTP. Para o conjunto de axiomas

$$e * x = x.$$
$$i(x) * x = e.$$
$$(x * y) * z = x * (y * z).$$
$$x * x = e.$$

pretende-se provar que

$$x * y = y * x.$$

Com a notação dos axiomas em TPTP, vem que

```
% Assumptions
fof(left_identity, axiom, (! [X] : m(e,X) = X)).
fof(left_inverse, axiom, (! [X] : m(i(X),X) = e)).
fof(associativity, axiom, (! [X,Y,Z] : m(m(X,Y),Z) = m(X, m(Y,Z)))).
fof(group_of_order_2, hypothesis, (! [X] : m(X,X) = e)).
% Conjecture
fof(commutativity, conjecture, (! [X,Y] : m(X,Y) = m(Y,X))).
```

O Vampire junta a negação do objetivo ao conjunto de pressupostos e tenta provar por refutação. A entrada TPTP foi submetida no Vampire. Um excerto do resultado é mostrado de seguida, em que se verifica que o Vampire devolve `Refutation found`.

```
% Refutation found. Thanks to Tanya!
% SZS status Theorem for SOT_S17X9R
% SZS output start Proof for SOT_S17X9R
fof(f317,plain,(
$false),
...
fof(f13,plain,(
( ! [X0 : $i,X1 : $i] : (m(X0,m(X0,X1)) = m(e,X1)) )),
inference(superposition,[],[f12,f9])).
fof(f12,plain,(
( ! [X2 : $i,X0 : $i,X1 : $i] : (m(m(X0,X1),X2) = m(X0,m(X1,X2))) )),
inference(cnf_transformation,[],[f3])).
...
fof(f6,negated_conjecture,(
```

[6] Vampire, an automatic theorem proving. `https://vprover.github.io`

```
~! [X0,X1] : m(X0,X1) = m(X1,X0)),
inference(negated_conjecture,[],[f5])).
fof(f5,conjecture,(
! [X0,X1] : m(X0,X1) = m(X1,X0)),
file('/tmp/IM14QLqEL8/SOT_Sl7X9R',commutativity)).
% SZS output end Proof for SOT_Sl7X9R
% ------------------------------
% Version: Vampire 4.6.0 (commit 0afb7ed4a on 2021-06-23 15:27:21 +0100)
% Termination reason: Refutation
```

2.2.2 Waldmeister

Waldmeister[7] é outro demonstrador de teoremas para LPO, sendo igualmente um dos demonstradores mais rápidos. O exemplo seguinte mostra a entrada no Waldmeister das cláusulas referidas no exemplo anterior.

```
NAME            waldmeister_example
MODE            PROOF
SORTS           ANY
SIGNATURE       e: -> ANY
                i: ANY -> ANY
                m: ANY ANY -> ANY
ORDERING        LPO
                i > m > e
VARIABLES       x,y,z : ANY
EQUATIONS       m(e,x) = x
                m(i(x),x) = e
                m(m(x,y),z) = m(x, m(y,z))
                m(x,x) = e
CONCLUSION      m(x,y) = m(y,x)
```

Um excerto do resultado é mostrado de seguida, em que o Waldmeister devolve `Goal proved`.

```
#START OF PROOF
0.2.0.0 : tes-eqn : m(X1,X1) = e : initial
0.2.1.0 : tes-eqn : m(X1,X1) = e : 0.2.0.0
0.2.2.0 : tes-rule : m(X1,X1) -> e : orient(0.2.1.0,u)
0.5.0.0 : tes-eqn : eq(X1,X1) = true : initial
0.5.1.0 : tes-eqn : eq(X1,X1) = true : 0.5.0.0
0.5.2.0 : tes-rule : eq(X1,X1) -> true : orient(0.5.1.0,u)
0.6.0.0 : tes-eqn : eq(m(X2,X1),m(X1,X2)) = false : initial
0.6.1.0 : tes-eqn : eq(m(X2,X1),m(X1,X2)) = false : 0.6.0.0
0.6.2.0 : tes-rule : eq(m(X1,X2),m(X2,X1)) -> false : orient(0.6.1.0,u)
0.8.0.0 : tes-eqn : false = eq(e,m(X1,X1)) : cp(0.6.2.0,L.1,0.2.2.0,L)
0.8.0.1 : tes-eqn : false = eq(e,e) : tes-red(0.8.0.0,R.2,0.2.2.0,L)
0.8.0.2 : tes-eqn : false = true : tes-red(0.8.0.1,R,0.5.2.0,L)
0.8.1.0 : tes-eqn : false = true : 0.8.0.2
0.8.2.0 : tes-rule : true -> false : orient(0.8.1.0,x)
1.0.0.0 : tes-goal : true = false : hypothesis
1.0.0.1 : tes-goal : false = false : tes-red(1.0.0.0,L,0.8.2.0,L)
1.0.0.2 : tes-final : false = false : 1.0.0.1
Proved Goals:
No.  1: true ?= false, current false ?= false
using narrowing to prove m(x2,x1) ?= m(x1,x2)
1 goal was specified, which was proved.
#END OF PROOF
...
Waldmeister states: Goal proved.
```

[7] Waldmeister, a theorem prover. https://www.mpi-inf.mpg.de/departments/automation-of-logic/software/waldmeister

2.2.3 Prover9

O demonstrador de teoremas escolhido nesta tese foi o Prover9[8]. É um demonstrador para LPO, que utiliza a resolução como principal regra de inferência. Uma vantagem do Prover9 é a de ter uma curva de aprendizagem menos acentuada relativamente a outros demonstradores, além do facto de a linguagem utilizada poder ser facilmente entendida por humanos [11]. Por todas estas razões é o sistema preferido pelos matemáticos que usam demonstração automática na sua investigação. Outra vantagem é a de ser acompanhado pelo programa Mace4[9], que tem como entrada o mesmo conjunto de axiomas, e executar em simultâneo a procura de um contra-exemplo. Assim, apesar de existirem muitos demonstradores automáticos de teoremas [9, 12], a escolha recaiu neste. Naturalmente, numa segunda fase, o sistema construído nesta tese poderá usar em *background* diferentes demonstradores e construtores de modelos de forma a tirar proveito da diferente rapidez em diferentes problemas.

Na secção 1.2 foi explicado o processo de resolução, onde é mostrado, como exemplo, que a *relação de inclusão entre conjuntos é transitiva*. No Prover9 pode ser definido como entrada:

```
formulas(assumptions).
(all x all y (S(x,y) <-> (all z (M(z,x) -> M(z,y))))).
end_of_list.

formulas(goals).
(all x all y all z (S(x,y) & S(y,z) -> S(x,z))).
end_of_list.
```

O mesmo conjunto de cláusulas é obtido para dar inicio ao processo de resolução:

```
3 S(x,y) | M(f1(x,y),x).
4 -S(x,y) | -M(z,x) | M(z,y).
5 S(x,y) | -M(f1(x,y),y).
6 S(c1,c2).
7 S(c2,c3).
8 -S(c1,c3).
```

Com as cláusulas 3 a 8, é obtida uma prova dado que o conjunto de cláusulas derivadas das premissas juntamente com a negação do conclusão, é o símbolo de cláusula vazia $F na etapa 18:

```
1 (all x all y (S(x,y) <-> (all z (M(z,x) -> M(z,y))))) # [assumption].
2 (all x all y all z (S(x,y) & S(y,z) -> S(x,z))) # [goal].
3 S(x,y) | M(f1(x,y),x).  [clausify(1)].
4 -S(x,y) | -M(z,x) | M(z,y).  [clausify(1)].
5 S(x,y) | -M(f1(x,y),y).  [clausify(1)].
6 S(c1,c2).  [deny(2)].
7 S(c2,c3).  [deny(2)].
8 -S(c1,c3).  [deny(2)].
11 -M(x,c1) | M(x,c2).  [resolve(6,a,4,a)].
12 -M(x,c2) | M(x,c3).  [resolve(7,a,4,a)].
13 M(f1(c1,c3),c1).  [resolve(8,a,3,a)].
14 -M(f1(c1,c3),c3).  [resolve(8,a,5,a)].
15 M(f1(c1,c3),c2).  [resolve(13,a,11,a)].
18 $F. [ur(12,b,14,a),unit_del(a,15)].
...
THEOREM PROVED
```

Por exemplo, a etapa 13 foi obtida pela resolução entre o primeiro termo da cláusula 8 e o primeiro termo da cláusula 3, expresso como $[resolve(8, a, 3, a)]$. Das cláusulas 8 $\{\neg S(c1, c3)\}$ e 3 $\{S(x, y)$,

[8] W. McCune. Prover9. version LADR-2009-11A. http://www.cs.unm.edu/~mccune/prover9/
[9] W. McCune. Mace4. version LADR-2009-11A. http://www.cs.unm.edu/~mccune/mace4/

$M(f_1(x,y), x)\}$ com $s = \{x/c1, y/c3\}$ segue que $\{\neg S(c1,c3)\}$, $\{S(c1,c3), M(f_1(c1,c3),c1)\}$ e então $\{M(f_1(c1, c3),c1)\}$.

O exemplo seguinte mostra a entrada no Prover9 das cláusulas usadas nos exemplos mostrados para o Vampire e para o Waldmeister.

```
formulas(assumptions).
m(e,x) = x.
m(i(x),x) = e.
m(m(x,y),z) = m(x, m(y,z)).
m(x,x) = e.
end_of_list.

formulas(goals).
m(x,y) = m(y,x).
end_of_list.
```

O que origina

```
1 m(x,y) = m(y,x) # label(non_clause) # label(goal).  [goal].
2 m(e,x) = x.  [assumption].
3 m(i(x),x) = e.  [assumption].
4 m(m(x,y),z) = m(x,m(y,z)).  [assumption].
5 m(x,x) = e.  [assumption].
6 m(c2,c1) != m(c1,c2).  [deny(1)].
7 m(i(x),m(x,y)) = y.  [para(3(a,1),4(a,1,1)),rewrite([2(2)]),flip(a)].
8 m(x,m(x,y)) = y.  [para(5(a,1),4(a,1,1)),rewrite([2(2)]),flip(a)].
9 m(x,m(y,m(x,y))) = e.  [para(5(a,1),4(a,1)),flip(a)].
11 m(i(i(x)),e) = x.  [para(3(a,1),7(a,1,2))].
13 m(i(x),e) = x.  [para(5(a,1),7(a,1,2))].
15 i(x) = x.  [back_rewrite(11),rewrite([13(4)])].
16 m(x,e) = x.  [back_rewrite(13),rewrite([15(1)])].
19 m(x,m(y,x)) = y.  [para(9(a,1),8(a,1,2)),rewrite([16(2)]),flip(a)].
24 m(x,y) = m(y,x).  [para(19(a,1),8(a,1,2))].
25 $F. [resolve(24,a,6,a)].
...
THEOREM PROVED
```

Normalmente o Prover9 é executado em paralelo com o gerador de modelos Mace4, em que se tenta encontrar um contra-exemplo para o mesmo conjunto de cláusulas de entrada do Prover9. Neste exemplo, o demonstrador de teoremas devolveu o resultado `theorem proved`, o que significa que não é possível encontrar um contra-exemplo e a sua execução em paralelo pode ser terminada. Segue-se uma breve descrição do funcionamento do Mace4 [16].

Para um dado domínio, todas as instâncias das *fbf* sobre o domínio são construídas pelo Mace4, sendo atribuída uma tabela para cada símbolo de função. Para um domínio de dimensão fixa n, é construída uma tabela $n \times n$ e os valores da tabela variam entre 0 e $n-1$. Se as cláusulas forem dependentes de outras, a atribuição de valores é propagada a outras cláusulas. Por exemplo, se $f(1,2) = 3$ é usado em $g(f(1,2)) = 4$ então tem-se $g(3) = 4$. Quando não se consegue atribuir o valor de verdade a uma célula da tabela, as atribuições que foram propagadas são desfeitas e são testadas outras atribuições. Se a todas as células de uma tabela for possível atribuir o valor de verdade, então foi encontrado um modelo, ou seja, uma interpretação para a qual todas as cláusulas de entrada são verdadeiras.

Normalmente tem-se que um demonstrador procura uma contradição para o conjunto de cláusulas $\{C_1, ..., C_k, \neg G\}$, enquanto que o Mace4 procura um modelo para o conjunto $\{C_1, ..., C_k, \neg G\}$. Se é encontrado um modelo, diz-se que se trata de um contra-exemplo, dado que não pode ocorrer simultaneamente G e $\neg G$.

Apesar de os demonstradores automáticos de teoremas existirem há várias décadas, a sua utilização pela comunidade matemática ainda está a dar os primeiros passos. Um estudo recente [8] mostra que uma das razões é a necessidade de melhoria de bibliotecas que sirvam de suporte no uso de sistemas de prova automática. Ainda que existam bibliotecas com milhares de definições e teoremas, são fundamentalmente usadas para testes de programas de prova ao invés de serem utilizadas na prática para potenciar o conhecimento. Este facto origina que, apesar de existirem ferramentas cada vez mais sofisticadas, faltam axiomáticas que possam ser usadas no trabalho de investigação matemática. Por outro lado, as axiomáticas disponíveis encontram-se dispersas na literatura ou em *sites* que não podem ser utilizados diretamente pelos sistemas de prova. Assim, um dos primeiros objetivos (e uma necessidade) desta tese foi a construção de um corpo de teorias e teoremas reais, usados em matemática. Esse foi o primeiro grande desafio que teve de ser vencido.

Capítulo 3

Solução proposta

O ponto de partida desta tese era a construção de uma base de dados de teorias e modelos.

Haver um localizador de modelos na literatura é importante pois quando um matemático encontra um modelo (eg, um contra-exemplo para a sua conjetura), talvez a literatura existente tenha alguma pista relevante sobre o modelo; ou quando um matemático identifica uma nova classe de álgebras pode gerar alguns modelos da sua teoria e procurar na literatura para tentar identificar ligações a outras áreas da matemática.

Mas o grande objetivo era ter o tal espaço de teorias-teoremas o que exigia extrair e codificar na linguagem do Prover9 muitas classes de álgebras e muitos teoremas sobre elas. Quanto mais rica for essa base de dados, mais forte seria a ferramenta de geração de matemática familiar, útil, relevante e nova que queríamos construir.

Para construir esta base de dados foram usados todos os artigos do arXiv[1] em termos que descrevemos de seguida, usando programas desenvolvidos para o efeito em Python[2].

3.1 Extração de tabelas

Foram extraídas centenas de tabelas de forma automática, a partir do código fonte dos artigos em Latex. Uma tabela é representada normalmente com dimensão $n \times n$, em que a primeira linha e coluna possuem os mesmos símbolos com uma mesma ordenação, não existem outros símbolos além dos usados nas linhas e colunas, etc. Símbolos diferentes entre tabelas mas com a mesma estrutura originaram tabelas iguais. Por exemplo, as duas tabelas seguintes mais à esquerda foram obtidas da literatura e posteriormente convertidas no formato da tabela à direita.

$\triangleright$	1	2	3	4
1	1	4	2	3
2	3	2	4	1
3	4	1	3	2
4	2	3	1	4

Ad	t	x	y	z
t	t	z	x	y
x	y	x	z	t
y	z	t	y	x
z	x	y	t	z

*	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

[1] ArXiv. Open access e-prints. Cornell University. `http://arxiv.org`
[2] Python 3. `http://www.python.org`

A tabela convertida é acompanhada das referências das outras duas tabelas que lhe deram origem:

`https://arxiv.org/pdf/1309.4634.pdf#page=5`
`https://arxiv.org/pdf/1101.0532.pdf#page=47`.

No final, foram obtidas 713 tabelas diferentes provenientes da literatura e 191 tabelas importadas de [13], totalizando 904 entradas de tabelas na base de dados.

3.1.1 Representante da classe de isomorfismo

Seja M um modelo representado por uma tabela de multiplicação de dimensão $n \times n$, e $\vec{v}(M)$ o vector de M formado pela concatenação de todas as linhas da tabela. Por exemplo, ao modelo M seguinte, corresponderá o vector $\vec{v}(M) = [0,3,1,2,2,1,3,0,3,0,2,\ 1,1,2,0,3]$.

*	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

Seja I o conjunto de modelos M_i isomorfos a M e $\vec{v}(M_i)$ o vector de M_i. Os vectores de M_i podem ser ordenados lexicograficamente. O primeiro vector da ordenação é considerado o modelo canónico, representante da classe de isomorfismo I.

Para o modelo M do exemplo, foram gerados todos os modelos isomorfos, tendo sido ordenados lexicograficamente e obtido o modelo canónico M':

*	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

dado que $\vec{v}(M') = [0,2,3,1,3,1,0,2,1,3,2,0,2,0,1,3]$ com

$\vec{v}(M') \leq_{lex} \vec{v}(M_i) \ldots \vec{v}(M) \cdots \leq_{lex} \vec{v}(M_j)$.

O algoritmo *representante da classe de isomorfismo* foi usado originalmente em [13, 14].

3.1.2 Imersão de modelos

Sejam μ_1 e μ_2 dois modelos, de ordem m_1 e m_2, com $m_1 < m_2$. Se existir um homomorfismo injetivo (também denominado imersão) de μ_1 e μ_2, então μ_1 é isomorfo a uma subálgebra de μ_2, ou seja, μ_1 *mergulha em* μ_2 e denota-se por $\mu_1 \lesssim \mu_2$.

A verificação de que um modelo mergulha em outro modelo pode ser verificada em Mace4. Seja, por exemplo, μ_1 de ordem 2 com a operação + definida por $\vec{v}(\mu_1) = [0,1,1,1]$ e μ_2 de ordem 3 com operação $*$ definida por $\vec{v}(\mu_2) = [0,1,2,1,1,2,2,2,2]$, tem-se que a entrada em Mace4 será:

```
if(Mace4).
assign(domain_size, 3).
assign(max_seconds, 10).
end_if.
formulas(assumptions).
% model order 2
% model #10
+(0,0)=0. +(0,1)=1.
+(1,0)=1. +(1,1)=1.
% model order 3
*(0,0)=0. *(0,1)=1. *(0,2)=2.
*(1,0)=1. *(1,1)=1. *(1,2)=2.
*(2,0)=2. *(2,1)=2. *(2,2)=2.
f(x) = f(y) -> x = y.
f(x + y) = f(x) * f(y).
end_of_list.
formulas(goals).
end_of_list.
```

3.1.3 Busca de modelos

A base de dados de modelos tem basicamente três funcionalidades de busca:

1. Procura um modelo (ou uma cópia isomorfa do modelo) fornecido pelo utilizador;
2. Procura modelos nos quais o modelo fornecido pelo utilizador mergulha;
3. Procura modelos que satisfazem uma axiomática dada.

Localizar na literatura um determinado modelo pode ser a ponte para informação relevante sobre o modelo ou para outras áreas da matemática. A primeira funcionalidade permite esta localização, sendo a segunda funcionalidade semelhante, apenas um pouco mais geral. Finalmente, sempre que um matemático define uma nova classe de álgebras, o primeiro passo é saber se a classe tem modelos não triviais e, caso os tenha, saber se tem modelos importantes. É pois muito útil descobrir modelos na literatura que satisfaçam propriedades dadas. Isso é feito pela terceira função de busca. Uma descrição pormenorizada da busca de modelos será feita no capítulo 5.

3.2 Extração de axiomáticas

A extração de axiomáticas (ou teorias, ou classes de álgebras) não foi feita de forma totalmente automática. Enquanto que na extração de tabelas é possível procurar por símbolos Latex que identificam tabelas, o reconhecimento automático de axiomáticas nos artigos requeria técnicas de processamento de linguagem natural fora do âmbito desta tese. No entanto, foram desenvolvidos programas em Python que identificavam automaticamente artigos com forte probabilidade de conter a definição axiomática de classes de álgebras (por exemplo, letras x, y, z acompanhadas com símbolos $*, +$). Selecionados os artigos, a codificação de dezenas de teorias foi feita manualmente, acompanhadas de muitos outros dados (como referências bibliográficas, etc.). No final, foram inseridos mais de cento e cinquenta classes de álgebras importantes (suficientemente importantes para ter artigos escritos sobre elas) no eixo de teorias da base de dados.

3.2.1 A base de dados de classes de álgebras

A lista completa das classes de álgebras inseridas em base de dados é descrita de seguida[3]. A base de dados de teorias, na sintaxe do Prover9, é apresentada no anexo 8.2.

#1 Almost distributive lattices
#2 BCI-algebras
#3 BCK-algebras
#4 BCK-join-semilattice
#5 BCK-lattices
#6 BCK-meet-semilattice
#7 Bilattice
#8 BL-algebras
#9 Boolean algebra
#10 Boolean monoid
#11 Boolean ring
#12 Bounded lattice
#13 Bounded residuated lattices
#14 Brouwerian algebras
#15 Brouwerian semilattices
#16 Chains
#17 Clifford algebras
#18 Commutative BCK-algebras
#19 Commutative integral ordered monoid
#20 Commutative lattice-ordered monoids
#21 Commutative lattice-ordered rings
#22 Commutative lattice-ordered semigroups
#23 Commutative ordered monoids
#24 Commutative ordered semigroups
#25 Commutative partially ordered monoids
#26 Commutative partially ordered semigroups
#27 Commutative regular rings
#28 Commutative residuated partially ordered monoids
#29 Complemented distributive lattices
#30 Complemented lattices
#31 Complemented modular lattices
#32 Loop
#33 De Morgan monoids
#34 Dense linear orders
#35 Directoids
#36 Distributive lattice ordered semigroups
#37 Division ring
#39 F-rings
#40 Fields
#41 FLcw-algebras
#42 Function rings
#43 Generalized Boolean algebras
#44 Hilbert algebras
#45 Hoop
#46 Implication algebras
#47 Implicative lattices
#48 Integral domains
#49 Integral ordered monoids
#50 Involutive lattices
#51 Join-semilattices
#52 Lattice-ordered pregroups
#53 Lattice-ordered rings
#54 Lattice-ordered semigroups
#55 Lineales
#56 Linear Heyting algebras
#57 Linear orders
#58 M-zeroids
#59 Medial quasigroups
#60 Meet-semilattices
#61 Modular ortholattices
#62 Monadic algebras
#63 Quasigroup
#64 Moufang quasigroups
#65 Multiplicative lattices
#66 Multiplicative semilattices
#67 Near-fields
#68 Near-ring
#69 Near-rings with identity
#70 Neardistributive lattices
#71 Normal bands
#72 Normal valued lattice-ordered groups
#73 Ockham algebras
#74 Order algebras
#75 Ordered monoids
#76 Ordered monoids with zero
#77 Ordered semigroup
#78 Ordered semilattice

[3]Uma vez que usamos muitas classes de objetos matemáticos, a maioria sem tradução oficial para português, optámos por usar o nome técnico inglês de forma a ser claro qual a estrutura em causa. De igual modo, quando nos referimos a um resultado na literatura (lema, proposição, teorema) usamos a classificação que consta no original (Lemma, Proposition, etc) de forma a garantir uma identificação clara do resultado.

#79 Ore domains
#80 Ortholattices
#81 Orthomodular lattices
#82 Partially ordered groups
#83 Partially ordered monoids
#84 Partially ordered semigroups
#85 Pocrims
#86 Posets
#87 Quandles
#88 Quasi-MV-algebra
#89 Quasi-ordered sets
#90 Residuated lattices
#91 Residuated partially ordered monoids
#92 Residuated posets
#93 Right hoops
#94 Semigroups with zero
#95 Semirings with identity
#96 Semirings with identity and zero
#97 Semirings with zero
#98 Sequential algebras
#99 Shells
#100 Skew lattices
#101 Skew-fields
#102 Tarski algebras
#103 Totally ordered monoids
#104 Wajsberg hoops
#105 Near-semiring
#106 MV-algebra
#107 Orthoimplication algebra
#108 Orthomodular implication algebra
#109 Quasi-implication algebra
#110 Idempotent semiring
#111 Concurrent semiring
#112 Kleene algebra
#113 Concurrent Kleene algebra
#114 Omega algebra
#115 Concurrent semiring with invariants
#116 Ground mereotopologies
#117 Domain semigroup
#118 Domain monoid
#119 Left closure semigroup
#120 Left closure monoid
#121 Inverse semigroup
#122 Domain range semigroup
#123 Antidomain monoid
#124 Semilattice pseudo-complemented semigroup
#125 Closable semilattice pseudo-complemented semigroup
#126 Cancellative semigroup
#127 Involutory quandle
#128 Left Bol loop
#129 Right Bol loop
#130 Left conjugacy closed loop
#131 Right conjugacy closed loop
#132 Conjugacy closed loop
#133 RIF loop
#134 ARIF loop
#135 Extra loop
#136 F-quasigroup
#137 Steiner quasigroup
#138 Steiner loop
#139 Rectangular quasigroup
#140 Rectangular loop
#141 Trisemigroup
#142 Trigroup
#144 Double Ward quasigroup
#145 Digroup
#146 Left disemigroup
#147 Right disemigroup
#148 Disemigroup
#149 Left I-Semiring
#150 Doppelsemigroup
#151 Strong doppelsemigroup
#152 Dimonoid
#153 Commutative dimonoid
#154 Diassociative semigroup
#155 Duplicial semigroup
#156 Extended duplicial semigroup
#157 Associative dialgebra
#158 Associative trialgebra
#159 Trialgebra
#160 Associative dimonoid

3.2.2 A representação interna das axiomáticas

Ao extrair teorias (ou axiomáticas) dos artigos, a primeira dificuldade é que a mesma classe de álgebras é muitas vezes representada com símbolos operacionais muito diferentes. Por exemplo, as

duas listas de fórmulas seguintes representam exatamente o mesmo teorema na mesma teoria (grupos em que cada elemento é inverso de si próprio são comutativos):

```
e * x = x.                          m(e, x) = x.
x' * x = e.                         m(i(x), x) = e.
((x * y) * z) = (x * (y * z)).      m(m(x, y), z) = m(x, m(y, z)).
(x * x) = e.                        m(x, x) = e.

x * y = y * x.                      m(x, y) = m(y, x).
```

Para evitar a multiplicação da mesma estrutura, foi criada uma forma canónica de representação interna (a menos da permutação de símbolos operacionais) que a seguir descrevemos. Faz-se uma aplicação injetiva do conjunto dos símbolos operacionais da teoria para o conjunto infinito $\{nullary1, nullary2, ..., unary1, unary2, ..., binary1, binayr2, ...\}$, função que preserva a aridade das funções e só tem imagem $n - ary_i$ se o símbolo $n - ary_{i-1}$ já pertence à imagem de f (para i>1). Além disso, todas as álgebras (na representação interna) são codificadas na forma prefixa.

A título de exemplo, o teorema de grupos referido acima, internamente é representado por:

```
formulas(assumptions).
binary1(nullary1,x) = x.
binary1(unary1(x),x) = nullary1.
binary1(binary1(x,y),z) = binary1(x,binary1(y,z)).
binary1(x,x) = nullary1.
end_of_list.

formulas(goals).
binary1(x,y) = binary1(y,x).
end_of_list.
```

O resultado será semelhante ao da listagem no final da secção 2.2.

```
1 binary1(x,y) = binary1(y,x) # label(non_clause) # label(goal). [goal].
2 binary1(nullary1,x) = x.  [assumption].
3 binary1(unary1(x),x) = nullary1.  [assumption].
4 binary1(binary1(x,y),z) = binary1(x,binary1(y,z)).  [assumption].
5 binary1(x,x) = nullary1.  [assumption].
6 binary1(c2,c1) != binary1(c1,c2).  [deny(1)].
7 binary1(unary1(x),binary1(x,y)) = y.  [para(3(a,1),4(a,1,1)),rewrite([2(2)]),flip(a)].
8 binary1(x,binary1(x,y)) = y.  [para(5(a,1),4(a,1,1)),rewrite([2(2)]),flip(a)].
9 binary1(x,binary1(y,binary1(x,y))) = nullary1.  [para(5(a,1),4(a,1)),flip(a)].
11 binary1(unary1(unary1(x)),nullary1) = x.  [para(3(a,1),7(a,1,2))].
13 binary1(unary1(x),nullary1) = x.  [para(5(a,1),7(a,1,2))].
15 unary1(x) = x.  [back_rewrite(11),rewrite([13(4)])].
16 binary1(x,nullary1) = x.  [back_rewrite(13),rewrite([15(1)])].
19 binary1(x,binary1(y,x)) = y.  [para(9(a,1),8(a,1,2)),rewrite([16(2)]),flip(a)].
24 binary1(x,y) = binary1(y,x).  [para(19(a,1),8(a,1,2))].
25 $F. [resolve(24,a,6,a)].
```

São ainda apresentados de seguida mais dois exemplos.

A classe definida pelos seguintes axiomas (Bilattice)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x + y) + z = x + (y + z).$$

$$x + y = y + x.$$

$$(x * y) * z = x * (y * z).$$

$$x * y = y * x.$$

$$(x + y) * x = x.$$

$$(x * y) + x = x.$$

$$\neg(x \vee y) = \neg x \wedge \neg y.$$

$$\neg\neg x = x.$$

$$\neg(x + y) = \neg x + \neg y.$$

$$\neg(x * y) = \neg x * \neg y.$$

é traduzida em:

```
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, y) = binary3(y, x).
binary4((binary4(x, y)), z) = binary4(x, (binary4(y, z))).
binary4(x, y) = binary4(y, x).
binary4((binary3(x, y)), x) = x.
binary3((binary4(x, y)), x) = x.
unary1(binary1(x, y)) = binary2(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
unary1(binary3(x, y)) = binary3(unary1(x), unary1(y)).
unary1(binary4(x, y)) = binary4(unary1(x), unary1(y)).
```

`https://marciedb.pythonanywhere.com/definition/7`

A função canónica entre símbolos é definida de seguida:

símbolos de predicado	símbolos de n-aridade
$\neg$	unary1
$\vee$	binary1
$\wedge$	binary2
$+$	binary3
$*$	binary4

A classe definida pelos seguintes axiomas (ordered semigroup)

$$(x * y) * z = x * (y * z).$$

$$x < x.$$

$$x < y \,\&\, y < x \rightarrow x = y.$$

$$x < y \,\&\, y < z \rightarrow x < z.$$

$$x < y \rightarrow ((x * z < y * z) \,\&\, (z * x < z * y)).$$

$$x < y \mid y < x.$$

é traduzida em:

```
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binaryr1(x, x).
binaryr1(x, y) & binaryr1(y, x) -> x = y.
binaryr1(x, y) & binaryr1(y, z) -> binaryr1(x, z).
binaryr1(x, y) -> binaryr1(binary1(x, z), binary1(y, z)) &
binaryr1(binary1(z, x), binary1(z, y)).
binaryr1(x, y) | binaryr1(y, x).
```

`https://marciedb.pythonanywhere.com/definition/77`

A função canónica entre símbolos é definida de seguida:

símbolos de predicado	símbolos de n-aridade
$*$	binary1
$<$	binaryr1

3.2.3 O conjunto parcialmente ordenado de teorias

O último passo na estruturação da base de dados de teorias consiste em criar o conjunto parcialmente ordenado das teorias.

Uma teoria T (conjunto de axiomas) diz-se menor que uma teoria Q, denotando-se $T \leq Q$, se T implica Q[4]. Dito de outra forma, $T \leq Q$ se todos os modelos que satisfazem T satisfazem também Q. Foi feito um varrimento exaustivo de todos os pares de teorias de forma a construir internamente o Diagrama de Hasse destas inclusões. Isto permite aceder diretamente a teorias extremas (que não estão contidas em nenhuma teoria ou que não contêm nenhuma teoria); permite ir percorrendo ramos de inclusões para ver qual a classe mais geral em que determinado teorema é verdadeiro, etc. Os maximais do conjunto parcialmente ordenado de teorias são apresentados de seguida.

#7 Bilattice
#10 Boolean monoid
#15 Brouwerian semilattices
#20 Commutative lattice-ordered monoids
#27 Commutative regular rings
#32 Loop
#47 Implicative lattices
#64 Moufang quasigroups
#104 Wajsberg hoops
#118 Domain monoid
#120 Left closure monoid
#123 Antidomain monoid
#135 Extra loop
#136 F-quasigroup
#142 Trigroup
#145 Digroup
#153 Commutative dimonoid
#156 Extended duplicial semigroup
#158 Associative trialgebra
#159 Trialgebra

[4]Na secção 4.5 são mostradas teorias T que implicam teorias Q.

3.3 Base de dados de teoremas

Neste eixo do nosso plano Teorias/Teoremas, temos essencialmente teoremas de semigrupos. Estes foram escolhidos por serem razoavelmente profundos e com provas intrincadas, à partida de lógica de primeira ordem ou facilmente transformáveis nisso, em muitos casos as classes nos quais os teoremas são provados são variedades ou quase-variedades, etc.

Temos um conjunto de dezenas de teoremas, descritos na secção 6.1, mas o sistema é escalável e poderá vir a ter milhares de teoremas das teorias mais diversas.

3.4 Arquitetura

Foram desenvolvidos inúmeros programas *batch* de extração e tratamento da informação. Foi usado o paradigma de programação orientada por objetos da linguagem Python, tendo permitido a reutilização de código usado nos programas *batch* e no código interno do *site*. Foi utilizada a framework Flask[5] escrita em Python para desenvolvimento do *site* e o motor de base de dados escolhido foi o SQLite[6]. A manutenção da base de dados é realizada através de programas *batch* que são executados em servidores externos. O alojamento *web* é efetuado em Pythonanywhere[7], contudo acessível via `www.marciex.com`.

Foi ainda necessário gerar os programas executáveis do Prover9 na máquina do alojamento. Na geração dos executáveis do Prover9 podem ocorrer erros de compilação, originando a não criação dos mesmos. O problema é resolvido com a seguinte alteração no ficheiro `Makefile` da pasta `LADR200911A/provers.src`: todas as ocorrências da opção `-lm` a seguir a `$(CFLAGS)` devem ser colocadas a seguir a `$(OBJECTS)`. Por exemplo, a linha

```
$(CC) $(CFLAGS) -lm -o prover9 prover9.o $(OBJECTS) ../ladr/libladr.a
```

deve ser alterada para

```
$(CC) $(CFLAGS) -o prover9 prover9.o $(OBJECTS) -lm ../ladr/libladr.a
```

Os sites que permitem o alojamento têm por norma recursos oferecidos por utilizador com bastantes limitações, de modo a atender às solicitações de todos os utilizadores de forma equitativa. É comum o Prover9 e o Mace4 utilizarem o CPU a 100% durante muito tempo, o que não é compatível com servidores partilhados. A maior parte dos algoritmos descritos origina o uso do CPU de forma exaustiva, levando o cancelamento dos *jobs* por parte dos sites de alojamento. O recurso à ferramenta CPULimit[8] permite o uso do CPU numa percentagem definida, por exemplo para 20%, o que origina um aumento significativo dos tempos de execução dos programas descritos, levando igualmente ao cancelamento dos *jobs* por parte dos sites de alojamento. Mesmo o alojamento num site com mais recursos como o Cloud VPS[9] não se apresentou como alternativa. Uma solução provisória passa por disponibilizar uma versão do programa para download e ser executado localmente no computador do utilizador. A solução definitiva terá de passar pela instalação num servidor dedicado em algum centro de investigação.

[5] Flask. Python web development framework. `https://palletsprojects.com/p/flask`
[6] SQLite. SQL database engine. `https://www.sqlite.org/index.html`
[7] Pythonanywhere. Python web hosting. `https://www.pythonanywhere.com`
[8] CPULimit. `https://github.com/opsengine/cpulimit`
[9] Cloud VPS. Hosting Services. `https://www.mvps.net`

Capítulo 4

A geração de novos teoremas

Uma vez construída a base de dados, estamos em condições de começar a produzir novos teoremas. De que tipo? Aqui a imaginação de cada um é o limite e é nesta parte que o virtuosismo do matemático do futuro se vai revelar. No que se segue fazemos algumas ilustrações apenas como prova de conceito.

Os teoremas que vamos gerar são essencialmente de dois tipos:

1. Teoremas que existem em alguma área da matemática (ie, enunciados bem definidos do tipo "se num grupo todo o elemento é inverso de si próprio, então o grupo é comutativo") e que tentamos exportar para outra área (ie, usa-se o mesmo enunciado, substituindo apenas a classe de álgebras considerada "se num semigrupo inverso todo o elemento é inverso de si próprio, então o semigrupo é comutativo"); neste tipo de teoremas é muito importante o Diagrama de Hasse construído na secção 3.2.3 pois só interesse tentar exportar o teorema para classe de álgebras que contêm aquela onde o teorema é válido, ou classes não relacionadas com ela.

2. Tipos de teoremas: neste caso não há um enunciado de um teorema concreto: existe um formato geral. A título de exemplo, é provar que uma variedade de álgebras definida por 4 axiomas pode ser definida por 3 axiomas. Outro exemplo, são os teoremas de subestrutura proibida como um reticulado é distributivo se e só se não contém o pentágono e o diamante. Neste caso não faz sentido substituir a classe de álgebras considerada por outra; o que faz sentido é procurar teoremas do tipo Um modelo M pertence à classe C se e só se pertence à classe B (que contém C) e o modelo N não pode ser mergulhado em M.

4.1 Teoremas geradores

Um dos propósitos desta tese é a exportação para outras áreas de teoremas de uma dada área. Por exemplo, em semigrupos com identidade é possível definir a relação de Green R: $\forall x, y \in S$,

$$x\mathcal{R}y \Leftrightarrow xS = yS.$$

Agora há muitos teoremas sobre esta relação e análogas. Mas a definição em si pode ser usada em qualquer classe de álgebras que contenha uma operação binária. Ou seja, a definição pode ser usada em praticamente todas as classes de álgebra ativamente usadas.

Por exemplo, consideremos a proposição 2.1.3 de [6]: *The relations $\mathcal{L}$ and $\mathcal{R}$ commute.* Esta proposição pode ser traduzida no formato Prover9 como

```
formulas(assumptions).
% Semigroup
(x * y) * z = x * (y * z).

% relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).

% relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.

formulas(goals).
% relations L and R commute
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

A submissão da listagem no Prover9 devolve `theorem proved`.

O nosso objetivo é exportar os resultados sobre conceitos de uma área para as outras áreas e ver em que casos os teoremas continuam válidos. Deste modo, percorre-se a base de dados de teorias e verifica-se para cada uma delas se o teorema considerado é válido. Seguem-se dois exemplos.

Na teoria *Commutative BCK-algebras*, o resultado obtido para a teoria *Semigroup* também é verdadeiro. A submissão da listagem seguinte no Prover9 também devolve `theorem proved`.

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).

% relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).

% relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.

formulas(goals).
% relations L and R commute
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

Na teoria *Directoids*, o resultado obtido anteriormente para a teoria *Semigroup* também é verdadeiro.

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.

% relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
```

```
% relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.

formulas(goals).
% relations L and R commute
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

O teorema obtido para a teoria *Semigroup* será denominado teorema gerador, enquanto os teoremas obtidos para as teorias *Commutative BCK-algebras* e *Directoids* serão denominados teoremas gerados.

Para cada teoria do anexo 8.2, foram gerados novos teoremas a partir dos teoremas da secção 6.1. Os resultados são mostrados na secção 6.2.

4.2 Redução do número de axiomas

Para um dado conjunto de n axiomas, o algoritmo de substituição procura obter um conjunto com $n-1$ axiomas, onde dois axiomas são substituídos por um axioma equivalente. A autoria do algoritmo é do Professor Michael Kinyon (University of Denver), onde o objectivo é obter um conjunto de $n-1$ axiomas equivalentes. Este algoritmo tira partido do modo de funcionamento do Prover9 e vai ser descrito de seguida. Como exemplo, vai ser aplicado o algoritmo de modo a que o seguinte conjunto de 4 axiomas *(Directoids)*

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

seja equivalente a um conjunto de 3 axiomas. As etapas do processo são as seguintes.

Etapa 1 Escolha das duas identidades i_1, i_2 que se pretende substituir por uma. São criados dois ficheiros de entrada para o Prover9 com cada identidade retirada do conjunto inicial e colocada em `goals`. Neste caso, foram escolhidas as identidades 2 e 4

```
formulas(assumptions).
x * x = x.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
end_of_list.
formulas(goals).
(x * y) * x = x * y.
end_of_list.

formulas(assumptions).
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
end_of_list.
formulas(goals).
```

```
x * ((x * y) * z) = (x * y) * z.
end_of_list.
```

Etapa 2 Obtenção de um modelo que satisfaça todas as identidades, menos a identidade i escolhida. Aplicando o Mace4 a cada um dos dois ficheiros, são obtidos dois modelos.

Etapa 3 Os modelos encontrados (ordem 2 e 3) são adicionados à lista `interpretations` e ao conjunto de axiomas inicial, com a opção do Prover9 `clear back demod`

```
clear(back_demod).
list(interpretations).
interpretation( 2, [number=1, seconds=0], [
function(c1, [ 0 ]), function(c2, [ 1 ]), function(*(_,_), [0, 1, 0, 1 ])]).
interpretation( 3, [number=1, seconds=0], [
function(c1, [ 0 ]), function(c2, [ 1 ]), function(c3, [ 2 ]),
function(*(_,_), [0, 1, 0, 1, 1, 2, 0, 2, 2 ]) ]).
end_of_list.
formulas(assumptions).
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
end_of_list.
```

O ficheiro criado serve de entrada ao Prover9, que gera um ficheiro com várias identidades f_i com a indicação `label(false)`

```
given #1 (I,wt=5): 1 x * x = x.  [assumption].
given #2 (I,wt=9): 2 (x * y) * x = x * y.  [assumption].
...
given #7 (F,wt=13): 57 ((x * y) * z) * x = (x * y) * z # label(false).
...
```

Etapa 4 Para cada identidade f_i, gerar dois ficheiros em que em `assumptions` tem f_i mais o conjunto de axiomas inicial sem i_1, i_2 e em `goals` tem, no primeiro ficheiro i_1 e no segundo ficheiro i_2. Ambos os ficheiros são executados sem a opção `clear back demod`. Se o Prover9 encontrar prova nos dois casos, então temos um teorema: os dois axiomas i_1 e i_2 podem ser substituídos por f_i; caso não encontre prova, mas encontre um contra-exemplo, esse é adicionado à lista de modelos existentes (Etapa 3) e volta-se a gerar e testar novas fórmulas (Etapas 3 e 4).

No final do processo descrito, tem-se que o conjunto de 4 axiomas *(Directoids)* é equivalente ao seguinte conjunto de 3 axiomas:

$$x * ((y * x) * z) = (y * x) * z.$$

$$x * x = x.$$

$$(x * y) * x = x * y.$$

Seguem-se alguns exemplos de teorias equivalentes com $n - 1$ axiomas.

Teorema 4.2.1. *O seguinte conjunto de axiomas (Directoids)*

$$x * x = x.$$

$$(x * y) * x = x * y.$$

$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

é equivalente a:

$$((x * y) * z) * x = (x * y) * z.$$
$$x * x = x.$$
$$y * (x * y) = x * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
end_of_list.
formulas(goals).
((x * y) * z) * x = (x * y) * z.
x * x = x.
y * (x * y) = x * y.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/35`. Substituição dos pares de axiomas 2 e 4. □

Teorema 4.2.2. *O seguinte conjunto de axiomas (Directoids)*

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

é equivalente a:

$$x * ((y * x) * z) = (y * x) * z.$$
$$x * x = x.$$
$$(x * y) * x = x * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
end_of_list.
formulas(goals).
x * ((y * x) * z) = (y * x) * z.
x * x = x.
(x * y) * x = x * y.
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/35. Substituição dos pares de axiomas 3 e 4. □

Teorema 4.2.3. *O seguinte conjunto de axiomas (Order algebras)*

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

é equivalente a:

$$(x * (y * x)) * x = y * x.$$
$$x * x = x.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
end_of_list.
formulas(goals).
(x * (y * x)) * x = y * x.
x * x = x.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/74. Substituição dos pares de axiomas 2 e 3. □

Teorema 4.2.4. *O seguinte conjunto de axiomas (Idempotent semiring)*

$$x + (y + z) = (x + y) + z.$$
$$x + y = y + x.$$
$$x + 0 = x.$$
$$0 + x = x.$$
$$x + x = x.$$
$$x * (y * z) = (x * y) * z.$$
$$x * 1 = x.$$

$$1 * x = x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

é equivalente a:

$$(1 * x) + (1 * y) = y + x.$$
$$x + (y + z) = (x + y) + z.$$
$$x + 0 = x.$$
$$0 + x = x.$$
$$x + x = x.$$
$$x * (y * z) = (x * y) * z.$$
$$x * 1 = x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x + (y + z) = (x + y) + z.
x + y = y + x.
x + 0 = x.
0 + x = x.
x + x = x.
x * (y * z) = (x * y) * z.
x * 1 = x.
1 * x = x.
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 0 = 0.
0 * x = 0.
end_of_list.
formulas(goals).
(1 * x) + (1 * y) = y + x.
x + (y + z) = (x + y) + z.
x + 0 = x.
0 + x = x.
x + x = x.
x * (y * z) = (x * y) * z.
x * 1 = x.
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 0 = 0.
0 * x = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/110`. Substituição dos pares de

axiomas 2 e 8. □

4.3 Localização de uma identidade

A localização de identidades é um instrumento genérico para procurar subestruturas relevantes de uma classe de álgebras. A ideia genérica é escolher um axioma da definição da classe de álgebras e substituir uma variável por uma constante, interpretando essa substituição como o conjunto dos elementos que satisfazem essa propriedade.

Vamos ilustrar o conceito com o exemplo dos semigrupos comutativos:

$$x(yz) = (xy)z \qquad xy = yx.$$

Localizar a segunda identidade significa fazer $ay = ya$, o que é interpretado como, dado um semigrupo S,

$$C := \{a \in S \mid ax = xa \ (\forall x \in S)\}.$$

Este conjunto é conhecido pelo *centro do semigrupo* S, ou seja, a localização da identidade $xy = yx$ nos semigrupos comutativos origina o centro de S.

É sabido que o centro de um semigrupo (caso seja não vazio) é um subsemigrupo. Na notação do Prover9:

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x * a = a * x.
x * b = b * x.
end_of_list.

formulas(goals).
x * (a * b) = (a * b) * x.
end_of_list.
```

Seguem-se alguns exemplos da localização de uma identidade.

Teorema 4.3.1. *Seja S a classe definida pelos seguintes axiomas (Inverse semigroup)*

$$(x * y) * z = x * (y * z).$$

$$x'' = x.$$

$$(x * x') * x = x.$$

$$(x * x') * (y * y') = (y * y') * (x * x').$$

A localização da quarta identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
% fix identity (x * x') * (y * y') = (y * y') * (x * x') and localize variable x
```

```
(a * a') * (y * y') = (y * y') * (a * a').
(b * b') * (y * y') = (y * y') * (b * b').
end_of_list.
formulas(goals).
((a * b) * (a * b)') * (y * y') = (y * y') * ((a * b) * (a * b)').
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/121. Max length proof: 95 □

Teorema 4.3.2. *Seja S a classe definida pelos seguintes axiomas (Normal bands)*

$$(x * y) * z = x * (y * z).$$

$$x * x = x.$$

$$((x * y) * z) * x = ((x * z) * y) * x.$$

A localização da terceira identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x * x = x.
% fix identity ((x * y) * z) * x = ((x * z) * y) * x and localize variable z
((x * y) * a) * x = ((x * a) * y) * x.
((x * y) * b) * x = ((x * b) * y) * x.
%
end_of_list.
formulas(goals).
((x * y) * (a*b)) * x = ((x * (a*b)) * y) * x.
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/71. Max length proof: 34 □

Teorema 4.3.3. *Seja S a classe definida pelos seguintes axiomas (Quandles)*

$$x \vee (y \vee z) = (x \vee y) \vee (x \vee z).$$

$$(x \wedge y) \wedge z = (x \wedge z) \wedge (y \wedge z).$$

$$(x \vee y) \wedge x = y.$$

$$x \vee (y \wedge x) = y.$$

$$x \vee x = x.$$

A localização da primeira identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% fix identity x v (y v z) = (x v y) v (x v z) and localize variable x
a v (y v z) = (a v y) v (a v z).
b v (y v z) = (b v y) v (b v z).
%
(x ^ y) ^ z = (x ^ z) ^ (y ^ z).
(x v y) ^ x = y.
x v (y ^ x) = y.
x v x = x.
end_of_list.
formulas(goals).
```

```
(a v b) v (y v z) = ((a v b) v y) v ((a v b) v z).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/87. Max length proof: 21 □

Teorema 4.3.4. *Seja S a classe definida pelos seguintes axiomas (Commutative regular rings)*

$$(x + y) + z = x + (y + z).$$

$$x + y = y + x.$$

$$x + 0 = x.$$

$$x + \neg x = 0.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$\exists\, y (x * y) * x = x.$$

$$x * y = y * x.$$

A localização da décima identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% fix identity exists y (x * y) * x = x and localize variable x
exists y (a * y) * a = a.
exists y (b * y) * b = b.
%
x * y = y * x.
end_of_list.
formulas(goals).
exists y ((a*b) * y) * (a*b) = (a*b).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/27. Max length proof: 20 □

Teorema 4.3.5. *Seja S a classe definida pelos seguintes axiomas (Order algebras)*

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

A localização da segunda identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x * x = x.
% fix identity (x * y) * x = y * x and localize variable x
(a * y) * a = y * a.
(b * y) * b = y * b.
%
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
end_of_list.
formulas(goals).
((a*b) * y) * (a*b) = y * (a*b).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/74`. Max length proof: 19 □

Teorema 4.3.6. *Seja S a classe definida pelos seguintes axiomas (Commutative BCK-algebras)*

$$((x * y) * (x * z)) * (z * y) = 0.$$
$$(x * (x * y)) * y = 0.$$
$$x * x = 0.$$
$$((x * y = 0) \ \& \ (y * x = 0)) \rightarrow (x = y).$$
$$(x * 0 = 0) \rightarrow (x = 0).$$
$$x * 0 = x.$$
$$x * (x * y) = y * (y * x).$$

A localização da sexta identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% fix identity  x * 0 = x and localize variable x
a * 0 = a.
b * 0 = b.
%
x * (x * y) = y * (y * x).
end_of_list.
formulas(goals).
(a*b) * 0 = (a*b).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/18`. Max length proof: 12 □

Teorema 4.3.7. *Seja S a classe definida pelos seguintes axiomas (Pocrims)*

$$((x * y) * (x * z)) * (z * y) = 0.$$
$$(x * (x * y)) * y = 0.$$
$$x * x = 0.$$
$$((x * y = 0)\ \&\ (y * x = 0)) \rightarrow (x = y).$$
$$(x * 0 = 0) \rightarrow (x = 0).$$
$$x * 0 = x.$$
$$(x * y) * z = x * (z \oplus y).$$

A localização da sexta identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% fix identity  x * 0 = x and localize variable x
a * 0 = a.
b * 0 = b.
%
(x * y) * z = x * (z @ y).
end_of_list.
formulas(goals).
(a*b) * 0 = (a*b).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/85`. Max length proof: 11 □

Teorema 4.3.8. *Seja S a classe definida pelos seguintes axiomas (Bilattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$(x + y) + z = x + (y + z).$$
$$x + y = y + x.$$
$$(x * y) * z = x * (y * z).$$
$$x * y = y * x.$$
$$(x + y) * x = x.$$
$$(x * y) + x = x.$$

$$\neg(x \vee y) = \neg x \wedge \neg y.$$
$$\neg\neg x = x.$$
$$\neg(x + y) = \neg x + \neg y.$$
$$\neg(x * y) = \neg x * \neg y.$$

A localização da décima sexta identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
-(x v y) = -x ^ -y.
--x = x.
-(x + y) = -x + -y.
% fix identity -(x * y) = -x * -y and localize variable x
-(a * y) = -a * -y.
-(b * y) = -b * -y.
%
end_of_list.
formulas(goals).
-((a*b) * y) = -(a*b) * -y.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/7`. Max length proof: 9 □

Teorema 4.3.9. *Seja S a classe definida pelos seguintes axiomas (Skew lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee x = x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge x = x.$$
$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

A localização da quarta identidade origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
(x v y) v z = x v (y v z).
x v x = x.
(x ^ y) ^ z = x ^ (y ^ z).
% fix identity x ^ x = x and localize variable x
a ^ a = a.
b ^ b = b.
```

```
%
x ^ (x v y) = x & x = x v (x ^ y).
(x v y) ^ y = y & y = (x ^ y) v y.
end_of_list.
formulas(goals).
(a^b) ^ (a^b) = (a^b).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/100. Max length proof: 9 □

Teorema 4.3.10. *Seja S a classe definida pelos seguintes axiomas (Antidomain monoid (a-monoid))*

$$(x * y) * z = x * (y * z).$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$x' * x = 0.$$
$$x * 0 = 0.$$
$$x' * y' = y' * x'.$$
$$x'' * x = x.$$
$$x' = (x * y)' * (x * y')'.$$
$$(x * y)' * x = ((x * y)' * x) * y'.$$

A localização da nona origina um reduto natural de subalgebra.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
% fix identity (x * y)' * x = ((x * y)' * x) * y' and localize variable x
(a * y)' * a = ((a * y)' * a) * y'.
(b * y)' * b = ((b * y)' * b) * y'.
end_of_list.
formulas(goals).
((a*b) * y)' * (a*b) = (((a*b) * y)' * (a*b)) * y'.
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/123. Max length proof: 4 □

4.4 Grafos

Seja G um grupo. O grafo de comutação (*commuting graph*) é definido do seguinte modo: para $g, h \in G$,

$$R(g, h) \leftrightarrow gh = hg.$$

Uma forma de ver este grafo é pensar em grupos comutativos:

$$(xy)z = x(yz) \qquad xe = x \qquad xx' = e \qquad xy = yx.$$

Agora mantemos todas as outras identidades usamos a comutatividade para definir um grafo, ou seja,

$$(xy)z = x(yz) \qquad xe = x \qquad xx' = e \qquad R(x,y) \Leftrightarrow xy = yx.$$

Este processo pode ser usado nas álgebras mais variadas definindo assim muitos grafos. Por exemplo, em semigrupos inversos:

$$(xy)z = x(yz) \quad xx'x = x \quad x'' = x \quad (xx')(y'y) = (y'y)(xx')$$

podemos aplicar a mesma ideia obtendo

$$(xy)z = x(yz) \quad xx'x = x \quad x'' = x \quad R(x,y) \Leftrightarrow (xx')(y'y) = (y'y)(xx').$$

Agora podemos provar relativamente à relação $R(x,y)$ muitos dos teoremas *standard* sobre grafos de comutação. Seguem-se algumas propostas, ilustradas com uma operação binária:

1. É $R(x,y)$ um grafo (ie, $R(x,y) \Rightarrow R(y,x)$)?
2. É $R(x,y)$ compatível com potências (ie, $R(x,y) \Rightarrow R(x*x, y*y)$)?
3. É $R(x,y)$ compatível com o produto à esquerda ou direita (ie, $R(x,y) \Rightarrow R(z*x, z*y)$)?
4. Será que o isomorfismo de grafos implica isomorfismo de grupos? (Por exemplo, será que existe uma bijeção entre dois grupos que preserva os grafos de comutação de ambos, então os grupos são isomorfos?)

Como exemplo, para semigrupos comutativos:

$$x(yz) = (xy)z \quad xy = yx$$

Escolhemos uma identidade (no caso a segunda) para definir uma relação binária:

$$R(x,y) \Leftrightarrow xy = yx.$$

Agora é necessário ver se R define um grafo (ie, se a relação é simétrica), e, em caso afirmativo, muitas perguntas podem ser feitas. Por exemplo:

```
formulas(assumptions).
(x * y) * z = x * (y * z).
% x * y = y * x.
R(x,y) <-> x * y = y * x.
R(a,b).
end_of_list.

formulas(goals).
R(*(a,a), *(b,b)).
end_of_list.
```

A submissão da listagem no Prover9 devolve `theorem proved` para $R(*(a,a), *(b,b))$.
Dois exemplos de grafos induzidos por classes de álgebras são mostrados de seguida.

Teorema 4.4.1. *Seja S a classe definida pelos seguintes axiomas (Steiner loop (Sloop))*

$$x * x = 1.$$
$$x * y = y * x.$$
$$x * (x * y) = y.$$

A terceira identidade induz um grafo natural.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x * x = 1.
x * y = y * x.
R(x,y) <-> x * (x * y) = y.
R(a,b).
end_of_list.
formulas(goals).
R(*(a,a), *(b,b)).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/138`. Max length proof: 7

□

Teorema 4.4.2. *Seja S a classe definida pelos seguintes axiomas (Steiner quasigroup (Squag))*

$$x * x = x.$$
$$x * y = y * x.$$
$$x * (x * y) = y.$$

A terceira identidade induz um grafo natural.

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x * x = x.
x * y = y * x.
R(x,y) <-> x * (x * y) = y.
R(a,b).
end_of_list.
formulas(goals).
R(*(a,a),*(b,b)).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/137`. Max length proof: 6

□

4.5 Inclusão de classes

Consideremos uma classe de álgebras C de um certo tipo τ, definida por um conjunto de axiomas A_C na linguagem de um conjunto de operações O, e outra classe de álgebras D do mesmo tipo

τ, definida por axiomas A_D na linguagem do mesmo conjunto de operações O. A classe C estará contida na classe D se existir uma prova de que A_C implica A_D.

De seguida são mostrados vários exemplos de inclusão de classes, após a redução à forma canónica das axiomáticas originais[1]. Foram obtidos teoremas na base de dados existentes, sendo que no seu conjunto formam o Diagrama de Hasse das axiomáticas, conforme descrito na secção 3.2.3. Os seguintes teoremas foram ordenados por ordem decrescente de `Max length proof`.

Teorema 4.5.1 (Extra loop implica ARIF loop)**.** *A classe definida pelos seguintes axiomas (Extra loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * (y * z)) * y = (x * y) * (z * y).$$

está contida na classe definida pelos seguintes axiomas (ARIF loop)

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(z * x) * ((y * x) * y) = (z * ((x * y) * x)) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, (binary1(y, z)))), y) =
binary1((binary1(x, y)), (binary1(z, y))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
```

[1] Uma vez que usamos muitas classes de objetos matemáticos, a maioria sem tradução oficial para português, optámos por usar o nome técnico inglês de forma a ser claro qual a estrutura em causa. De igual modo, quando nos referimos a um resultado na literatura (lema, proposição, teorema) usamos a classificação que consta no original (Lemma, Proposition, etc) de forma a garantir uma identificação clara do resultado.

```
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(z, x)), (binary1((binary1(y, x)), y))) =
binary1((binary1(z, (binary1((binary1(x, y)), x)))), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/135`, definition #135 implies definition #134. Number of proofs: 7. Max length proof: 141. □

Teorema 4.5.2 (Extra loop implica Conjugacy closed loop)**.** *A classe definida pelos seguintes axiomas (Extra loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * (y * z)) * y = (x * y) * (z * y).$$

está contida na classe definida pelos seguintes axiomas (Conjugacy closed loop)

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$z * (y * x) = ((z * y) \,/\, z) * (z * x).$$
$$(x * y) * z = (x * z) * (z \setminus (y * z)).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, (binary1(y, z)))), y) =
binary1((binary1(x, y)), (binary1(z, y))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
```

```
binary1(x, 1) = x.
x = binary1(1, x).
binary1(z, (binary1(y, x))) =
binary1((binary3((binary1(z, y)), z)), (binary1(z, x))).
binary1((binary1(x, y)), z) =
binary1((binary1(x, z)), (binary2(z, (binary1(y, z))))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/135`, definition #135 implies definition #132. Number of proofs: 8. Max length proof: 116. □

Teorema 4.5.3 (Extra loop implica Left conjugacy closed loop)**.** *A classe definida pelos seguintes axiomas (Extra loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * (y * z)) * y = (x * y) * (z * y).$$

está contida na classe definida pelos seguintes axiomas (Left conjugacy closed loop)

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$z * (y * x) = ((z * y) \,/\, z) * (z * x).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, (binary1(y, z)))), y) =
binary1((binary1(x, y)), (binary1(z, y))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
```

```
binary1(z, (binary1(y, x))) =
binary1((binary3((binary1(z, y)), z)), (binary1(z, x))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/135`, definition #135 implies definition #130. Number of proofs: 7. Max length proof: 116. □

Teorema 4.5.4 (Extra loop implica RIF loop)**.** *A classe definida pelos seguintes axiomas (Extra loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * (y * z)) * y = (x * y) * (z * y).$$

está contida na classe definida pelos seguintes axiomas (RIF loop)

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * y) * (z * (x * y)) = ((x * (y * z)) * x) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, (binary1(y, z)))), y) =
binary1((binary1(x, y)), (binary1(z, y))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, y)), (binary1(z, (binary1(x, y))))) =
binary1((binary1((binary1(x, (binary1(y, z)))), x)), y).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/135, definition #135 implies definition #133. Number of proofs: 7. Max length proof: 104. □

Teorema 4.5.5 (Extra loop implica Right Bol loop)**.** *A classe definida pelos seguintes axiomas (Extra loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * (y * z)) * y = (x * y) * (z * y).$$

está contida na classe definida pelos seguintes axiomas (Right Bol loop)

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$z * ((x * y) * x) = ((z * x) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, (binary1(y, z)))), y) =
binary1((binary1(x, y)), (binary1(z, y))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1(z, (binary1((binary1(x, y)), x))) =
binary1((binary1((binary1(z, x)), y)), x).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/135, definition #135 implies definition #129. Number of proofs: 7. Max length proof: 83. □

Teorema 4.5.6 (Boolean algebra implica MV-algebra)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (MV-algebra)

$$x + y = y + x.$$
$$(x + y) + z = x + (y + z).$$
$$x + x' = 1.$$
$$x + 0 = x.$$
$$(x')' = x.$$
$$0' = 1.$$
$$x + 1 = 1.$$
$$(x' + y)' + y = (x + y')' + x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1(x, y) = binary1(y, x).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, unary1(x)) = 1.
binary1(x, 0) = x.
```

```
unary1(unary1(x)) = x.
unary1(0) = 1.
binary1(x, 1) = 1.
binary1((unary1(binary1(unary1(x), y))), y) =
binary1((unary1(binary1(x, unary1(y)))), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #106. Number of proofs: 8. Max length proof: 58. □

Teorema 4.5.7 (Boolean algebra implica Quasi-MV-algebra)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Quasi-MV-algebra)

$$(x + y) + z = x + (y + z).$$
$$x'' = x.$$
$$x + 1 = 1.$$
$$(x' + y)' + y = (y' + x)' + x.$$
$$(x + 0)' = x' + 0.$$
$$(x + 0) + 0 = x + 0.$$
$$0' = 1.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
```

```
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1(x, 1) = 1.
binary1((unary1(binary1(unary1(x), y))), y) =
binary1((unary1(binary1(unary1(y), x))), x).
(unary1(binary1(x, 0))) = binary1(unary1(x), 0).
binary1((binary1(x, 0)), 0) = binary1(x, 0).
unary1(0) = 1.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #88. Number of proofs: 7. Max length proof: 58. □

Teorema 4.5.8 (Boolean monoid implica MV-algebra)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$
$$(x * y) * z = x * (y * z).$$
$$e * x = x.$$
$$x * e = x.$$
$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$
$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (MV-algebra)

$$x + y = y + x.$$
$$(x + y) + z = x + (y + z).$$
$$x + x' = 1.$$
$$x + 0 = x.$$

$$(x')' = x.$$

$$0' = 1.$$

$$x + 1 = 1.$$

$$(x' + y)' + y = (x + y')' + x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1(x, y) = binary1(y, x).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, unary1(x)) = 1.
binary1(x, 0) = x.
unary1(unary1(x)) = x.
unary1(0) = 1.
binary1(x, 1) = 1.
binary1((unary1(binary1(unary1(x), y))), y) =
binary1((unary1(binary1(x, unary1(y)))), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #106. Number of proofs: 8. Max length proof: 58. □

Teorema 4.5.9 (Boolean monoid implica Quasi-MV-algebra)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Quasi-MV-algebra)

$$(x + y) + z = x + (y + z).$$

$$x'' = x.$$

$$x + 1 = 1.$$

$$(x' + y)' + y = (y' + x)' + x.$$

$$(x + 0)' = x' + 0.$$

$$(x + 0) + 0 = x + 0.$$

$$0' = 1.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1(x, 1) = 1.
binary1((unary1(binary1(unary1(x), y))), y) =
binary1((unary1(binary1(unary1(y), x))), x).
(unary1(binary1(x, 0))) = binary1(unary1(x), 0).
binary1((binary1(x, 0)), 0) = binary1(x, 0).
unary1(0) = 1.
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/10, definition #10 implies definition #88. Number of proofs: 7. Max length proof: 58. □

Teorema 4.5.10 (Complemented distributive lattices implica MV-algebra)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (MV-algebra)

$$x + y = y + x.$$
$$(x + y) + z = x + (y + z).$$
$$x + x' = 1.$$
$$x + 0 = x.$$
$$(x')' = x.$$
$$0' = 1.$$
$$x + 1 = 1.$$
$$(x' + y)' + y = (x + y')' + x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
```

```
binary1(x, y) = binary1(y, x).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, unary1(x)) = 1.
binary1(x, 0) = x.
unary1(unary1(x)) = x.
unary1(0) = 1.
binary1(x, 1) = 1.
binary1((unary1(binary1(unary1(x), y))), y) =
binary1((unary1(binary1(x, unary1(y)))), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #106. Number of proofs: 8. Max length proof: 58. □

Teorema 4.5.11 (Complemented distributive lattices implica Quasi-MV-algebra)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Quasi-MV-algebra)

$$(x + y) + z = x + (y + z).$$
$$x'' = x.$$
$$x + 1 = 1.$$
$$(x' + y)' + y = (y' + x)' + x.$$
$$(x + 0)' = x' + 0.$$
$$(x + 0) + 0 = x + 0.$$
$$0' = 1.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
```

```
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1(x, 1) = 1.
binary1((unary1(binary1(unary1(x), y))), y) =
binary1((unary1(binary1(unary1(y), x))), x).
(unary1(binary1(x, 0))) = binary1(unary1(x), 0).
binary1((binary1(x, 0)), 0) = binary1(x, 0).
unary1(0) = 1.
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/29, definition #29 implies definition #88. Number of proofs: 7. Max length proof: 58. □

Teorema 4.5.12 (Boolean algebra implica Ockham algebras)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Ockham algebras)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$

$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #73. Number of proofs: 13. Max length proof: 57. □

Teorema 4.5.13 (Boolean algebra implica Orthomodular lattices)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Orthomodular lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$
$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
binary1(x, (binary2(unary1(x), (binary1(x, y))))) = binary1(x, y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #81. Number of proofs: 13. Max length proof: 57. □

Teorema 4.5.14 (Boolean algebra implica Ortholattices)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Ortholattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
```

```
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #80. Number of proofs: 12. Max length proof: 57. □

Teorema 4.5.15 (Boolean monoid implica Ockham algebras)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Ockham algebras)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$(x \wedge y)' = x' \vee y'.$$

$$(x \vee y)' = x' \wedge y'.$$

$$0' = 1.$$

$$1' = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition

#73. Number of proofs: 13. Max length proof: 57. □

Teorema 4.5.16 (Boolean monoid implica Orthomodular lattices)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$
$$(x * y) * z = x * (y * z).$$
$$e * x = x.$$
$$x * e = x.$$
$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$
$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Orthomodular lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$

$$(x \wedge y)' = x' \vee y'.$$

$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
binary1(x, (binary2(unary1(x), (binary1(x, y))))) = binary1(x, y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #81. Number of proofs: 13. Max length proof: 57. □

Teorema 4.5.17 (Boolean monoid implica Ortholattices)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$
$$(x * y) * z = x * (y * z).$$
$$e * x = x.$$
$$x * e = x.$$
$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$
$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Ortholattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
```

```
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #80. Number of proofs: 12. Max length proof: 57. □

Teorema 4.5.18 (Complemented distributive lattices implica Ockham algebras)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Ockham algebras)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #73. Number of proofs: 13. Max length proof: 57. □

Teorema 4.5.19 (Complemented distributive lattices implica Orthomodular lattices)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Orthomodular lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$
$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
binary1(x, (binary2(unary1(x), (binary1(x, y))))) = binary1(x, y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #81. Number of proofs: 13. Max length proof: 57. □

Teorema 4.5.20 (Complemented distributive lattices implica Ortholattices)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Ortholattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
```

```
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #80. Number of proofs: 12. Max length proof: 57. □

Teorema 4.5.21 (Boolean algebra implica Involutive lattices)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Involutive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$\neg(x \wedge y) = \neg x \vee \neg y.$$

$$\neg\neg x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
unary1(binary2(x, y)) = binary1(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #50. Number of proofs: 8. Max length proof: 53. □

Teorema 4.5.22 (Boolean monoid implica Involutive lattices)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \, (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \quad (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Involutive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$\neg(x \wedge y) = \neg x \vee \neg y.$$
$$\neg\neg x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
unary1(binary2(x, y)) = binary1(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #50. Number of proofs: 8. Max length proof: 53. □

Teorema 4.5.23 (Complemented distributive lattices implica Involutive lattices)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Involutive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$\neg(x \wedge y) = \neg x \vee \neg y.$$
$$\neg\neg x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
unary1(binary2(x, y)) = binary1(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #50. Number of proofs: 8. Max length proof: 53. □

Teorema 4.5.24 (Extra loop implica Left Bol loop). *A classe definida pelos seguintes axiomas (Extra loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * (y * z)) * y = (x * y) * (z * y).$$

está contida na classe definida pelos seguintes axiomas (Left Bol loop)

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$x * (y * (x * z)) = (x * (y * x)) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, (binary1(y, z)))), y) =
binary1((binary1(x, y)), (binary1(z, y))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1(x, (binary1(y, (binary1(x, z))))) =
binary1((binary1(x, (binary1(y, x)))), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/135`, definition #135 implies definition #128. Number of proofs: 7. Max length proof: 50. □

Teorema 4.5.25 (Complemented distributive lattices implica Closable semilattice pseudo-complemented semigroup). *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Closable semilattice pseudo-complemented semigroup)

$$(x * y) * z = x * (y * z).$$
$$x' * x = 0.$$
$$x * 0' = x.$$
$$x' * y' = y' * x'.$$
$$x' * (x' * y)' = x' * y'.$$
$$x'' * x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
binary1((binary2(x, y)), x) = x.
binary2((binary1(x, y)), x) = x.
binary2(0, x) = x.
binary2(1, x) = 1.
binary1(x, (binary2(y, z))) = binary2((binary1(x, y)), (binary1(x, z))).
binary2(x, unary1(x)) = 1.
binary1(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(unary1(x), x) = 0.
binary1(x, unary1(0)) = x.
binary1(unary1(x), unary1(y)) = binary1(unary1(y), unary1(x)).
binary1(unary1(x), (unary1(binary1(unary1(x), y)))) =
binary1(unary1(x), unary1(y)).
binary1(unary1(unary1(x)), x) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition

#125. Number of proofs: 6. Max length proof: 43. □

Teorema 4.5.26 (Boolean ring implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Boolean ring)*

$$(x * y) * z = x * (y * z).$$
$$x * y = y * x.$$
$$x * 0 = x.$$
$$0 * x = x.$$
$$x * x = 0.$$
$$(x \circ y) \circ z = x \circ (y \circ z).$$
$$1 \circ x = x.$$
$$x \circ 1 = x.$$
$$x \circ x = x.$$
$$x \circ (y * z) = (x \circ y) * (x \circ z).$$
$$(x * y) \circ z = (x \circ z) * (y \circ z).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2(x, 0) = x.
binary2(0, x) = x.
binary2(x, x) = 0.
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(1, x) = x.
binary1(x, 1) = x.
binary1(x, x) = x.
binary1(x, (binary2(y, z))) = binary2((binary1(x, y)), (binary1(x, z))).
binary1((binary2(x, y)), z) = binary2((binary1(x, z)), (binary1(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/11`, definition #11 implies definition #71. Number of proofs: 3. Max length proof: 31. □

Teorema 4.5.27 (Implication algebras implica Quasi-implication algebra)**.** *A classe definida pelos seguintes axiomas (Implication algebras)*

$$(x \circ y) \circ x = x.$$
$$(x \circ y) \circ y = (y \circ x) \circ x.$$
$$x \circ (y \circ z) = y \circ (x \circ z).$$

está contida na classe definida pelos seguintes axiomas (Quasi-implication algebra)

$$(x * y) * x = x.$$
$$(x * y) * (x * z) = (y * x) * (y * z).$$
$$((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), x) = x.
binary1((binary1(x, y)), y) = binary1((binary1(y, x)), x).
binary1(x, (binary1(y, z))) = binary1(y, (binary1(x, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), x) = x.
binary1((binary1(x, y)), (binary1(x, z))) =
binary1((binary1(y, x)), (binary1(y, z))).
binary1((binary1((binary1(x, y)), (binary1(y, x)))), x) =
binary1((binary1((binary1(y, x)), (binary1(x, y)))), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/46`, definition #46 implies definition #109. Number of proofs: 3. Max length proof: 25. □

Teorema 4.5.28 (Tarski algebras implica Quasi-implication algebra)**.** *A classe definida pelos seguintes axiomas (Tarski algebras)*

$$(x \circ y) \circ x = x.$$
$$(x \circ y) \circ y = (y \circ x) \circ x.$$
$$x \circ (y \circ z) = y \circ (x \circ z).$$

está contida na classe definida pelos seguintes axiomas (Quasi-implication algebra)

$$(x * y) * x = x.$$
$$(x * y) * (x * z) = (y * x) * (y * z).$$
$$((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), x) = x.
binary1((binary1(x, y)), y) = binary1((binary1(y, x)), x).
binary1(x, (binary1(y, z))) = binary1(y, (binary1(x, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), x) = x.
binary1((binary1(x, y)), (binary1(x, z))) =
binary1((binary1(y, x)), (binary1(y, z))).
```

```
binary1((binary1((binary1(x, y)), (binary1(y, x)))), x) =
binary1((binary1((binary1(y, x)), (binary1(x, y)))), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/102`, definition #102 implies definition #109. Number of proofs: 3. Max length proof: 25. □

Teorema 4.5.29 (Ockham algebras implica Neardistributive lattices)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Neardistributive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
```

```
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y)))))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y)))))))))).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/73, definition #73 implies definition #70. Number of proofs: 8. Max length proof: 23. □

Teorema 4.5.30 (Boolean algebra implica Neardistributive lattices)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Neardistributive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y)))))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y)))))))))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #70. Number of proofs: 8. Max length proof: 22. □

Teorema 4.5.31 (Boolean monoid implica Neardistributive lattices)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$
$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Neardistributive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$

$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y)))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y)))))))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #70. Number of proofs: 8. Max length proof: 22. □

Teorema 4.5.32 (Complemented distributive lattices implica Neardistributive lattices)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Neardistributive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
```

```
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y))))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y))))))))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #70. Number of proofs: 8. Max length proof: 22. □

Teorema 4.5.33 (Distributive lattice ordered semigroups implica Neardistributive lattices)**.** *A classe definida pelos seguintes axiomas (Distributive lattice ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x * y) * z = x * (y * z).$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Neardistributive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
```

```
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y)))))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y)))))))))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/36`, definition #36 implies definition #70. Number of proofs: 8. Max length proof: 21. □

Teorema 4.5.34 (Ockham algebras implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #74. Number of proofs: 5. Max length proof: 21. □

Teorema 4.5.35 (Ockham algebras implica Directoids)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$

$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #35. Number of proofs: 4. Max length proof: 21. □

Teorema 4.5.36 (Boolean algebra implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$

$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #74. Number of proofs: 5. Max length proof: 20. □

Teorema 4.5.37 (Boolean algebra implica Directoids)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$

$$(x * y) * x = x * y.$$

$$y * (x * y) = x * y.$$

$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #35. Number of proofs: 4. Max length proof: 20. □

Teorema 4.5.38 (Boolean monoid implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$
$$(x * y) * z = x * (y * z).$$
$$e * x = x.$$
$$x * e = x.$$
$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$
$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$

$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #74. Number of proofs: 5. Max length proof: 20. □

Teorema 4.5.39 (Boolean monoid implica Directoids)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$

$$(x * y) * x = x * y.$$

$$y * (x * y) = x * y.$$

$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #35. Number of proofs: 4. Max length proof: 20. □

Teorema 4.5.40 (Complemented distributive lattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #74. Number of proofs: 5. Max length proof: 20. □

Teorema 4.5.41 (Complemented distributive lattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #35. Number of proofs: 4. Max length proof: 20. □

Teorema 4.5.42 (Ockham algebras implica Idempotent semiring)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Idempotent semiring)

$$x + (y + z) = (x + y) + z.$$
$$x + y = y + x.$$
$$x + 0 = x.$$
$$0 + x = x.$$
$$x + x = x.$$
$$x * (y * z) = (x * y) * z.$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1(x, (binary1(y, z))) = binary1((binary1(x, y)), z).
binary1(x, y) = binary1(y, x).
```

```
binary1(x, 0) = x.
binary1(0, x) = x.
binary1(x, x) = x.
binary2(x, (binary2(y, z))) = binary2((binary2(x, y)), z).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #110. Number of proofs: 12. Max length proof: 20. □

Teorema 4.5.43 (Ockham algebras implica Semirings with identity and zero)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity and zero)

$$(x * y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #96. Number of proofs: 8. Max length proof: 20. □

Teorema 4.5.44 (Ockham algebras implica Semirings with identity)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity)

$$(x * y) * z = x * (y * z).$$

$$x + y = y + x.$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #95. Number of proofs: 6. Max length proof: 20. □

Teorema 4.5.45 (Ockham algebras implica Shells)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$(x \wedge y)' = x' \vee y'.$$

$$(x \vee y)' = x' \wedge y'.$$

$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$
$$x + 0 = x.$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$0 * x = 0.$$
$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #99. Number of proofs: 6. Max length proof: 20. □

Teorema 4.5.46 (Boolean algebra implica Idempotent semiring)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$

$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Idempotent semiring)

$$x + (y + z) = (x + y) + z.$$
$$x + y = y + x.$$
$$x + 0 = x.$$
$$0 + x = x.$$
$$x + x = x.$$
$$x * (y * z) = (x * y) * z.$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1(x, (binary1(y, z))) = binary1((binary1(x, y)), z).
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(0, x) = x.
binary1(x, x) = x.
binary2(x, (binary2(y, z))) = binary2((binary2(x, y)), z).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition

#110. Number of proofs: 12. Max length proof: 19. □

Teorema 4.5.47 (Boolean algebra implica Semirings with identity and zero)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity and zero)

$$(x * y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
```

```
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #96. Number of proofs: 8. Max length proof: 19. □

Teorema 4.5.48 (Boolean algebra implica Semirings with identity)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity)

$$(x * y) * z = x * (y * z).$$

$$x + y = y + x.$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
```

```
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #95. Number of proofs: 6. Max length proof: 19. □

Teorema 4.5.49 (Boolean algebra implica Shells)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$
$$x + 0 = x.$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$0 * x = 0.$$
$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
```

```
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #99. Number of proofs: 6. Max length proof: 19. □

Teorema 4.5.50 (Boolean monoid implica Idempotent semiring)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Idempotent semiring)

$$x + (y + z) = (x + y) + z.$$

$$x + y = y + x.$$

$$x + 0 = x.$$

$$0 + x = x.$$

$$x + x = x.$$

$$x * (y * z) = (x * y) * z.$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 0 = 0.$$

$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1(x, (binary1(y, z))) = binary1((binary1(x, y)), z).
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(0, x) = x.
binary1(x, x) = x.
binary2(x, (binary2(y, z))) = binary2((binary2(x, y)), z).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #110. Number of proofs: 12. Max length proof: 19. □

Teorema 4.5.51 (Boolean monoid implica Semirings with identity and zero)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$
$$(x * y) * z = x * (y * z).$$
$$e * x = x.$$
$$x * e = x.$$
$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$
$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity and zero)

$$(x * y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
```

```
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #96. Number of proofs: 8. Max length proof: 19. □

Teorema 4.5.52 (Boolean monoid implica Semirings with identity)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity)

$$(x * y) * z = x * (y * z).$$

$$x + y = y + x.$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #95. Number of proofs: 6. Max length proof: 19. □

Teorema 4.5.53 (Boolean monoid implica Shells)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$

$$x + 0 = x.$$

$$1 * x = x.$$

$$x * 1 = x.$$

$$0 * x = 0.$$

$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #99. Number of proofs: 6. Max length proof: 19. □

Teorema 4.5.54 (Complemented distributive lattices implica Idempotent semiring)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Idempotent semiring)

$$x + (y + z) = (x + y) + z.$$
$$x + y = y + x.$$
$$x + 0 = x.$$
$$0 + x = x.$$
$$x + x = x.$$
$$x * (y * z) = (x * y) * z.$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1(x, (binary1(y, z))) = binary1((binary1(x, y)), z).
binary1(x, y) = binary1(y, x).
```

```
binary1(x, 0) = x.
binary1(0, x) = x.
binary1(x, x) = x.
binary2(x, (binary2(y, z))) = binary2((binary2(x, y)), z).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #110. Number of proofs: 12. Max length proof: 19. □

Teorema 4.5.55 (Complemented distributive lattices implica Semirings with identity and zero)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity and zero)

$$(x * y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
```

```
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #96. Number of proofs: 8. Max length proof: 19. □

Teorema 4.5.56 (Complemented distributive lattices implica Semirings with identity)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity)

$$(x * y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #95. Number of proofs: 6. Max length proof: 19. □

Teorema 4.5.57 (Complemented distributive lattices implica Shells)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$
$$x + 0 = x.$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$0 * x = 0.$$

$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #99. Number of proofs: 6. Max length proof: 19. □

Teorema 4.5.58 (Distributive lattice ordered semigroups implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Distributive lattice ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$(x * y) * z = x * (y * z).$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$

$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/36`, definition #36 implies definition #74. Number of proofs: 5. Max length proof: 19. □

Teorema 4.5.59 (Distributive lattice ordered semigroups implica Directoids)**.** *A classe definida pelos seguintes axiomas (Distributive lattice ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x * y) * z = x * (y * z).$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
```

```
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/36`, definition #36 implies definition #35. Number of proofs: 4. Max length proof: 19. □

Teorema 4.5.60 (Implicative lattices implica Neardistributive lattices)**.** *A classe definida pelos seguintes axiomas (Implicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \circ (y \vee z) = (x \circ y) \vee (x \circ z).$$
$$x \circ (y \wedge z) = (x \circ y) \wedge (x \circ z).$$
$$(x \vee y) \circ z = (x \circ z) \wedge (y \circ z).$$
$$(x \wedge y) \circ z = (x \circ z) \vee (y \circ z).$$

está contida na classe definida pelos seguintes axiomas (Neardistributive lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3(x, (binary2(y, z))) = binary2((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary2((binary3(x, z)), (binary3(y, z))).
binary3((binary2(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y)))))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y)))))))))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/47`, definition #47 implies definition #70. Number of proofs: 8. Max length proof: 19. □

Teorema 4.5.61 (Antidomain monoid implica Closable semilattice pseudo-complemented semigroup)**.** *A classe definida pelos seguintes axiomas (Antidomain monoid)*

$$(x * y) * z = x * (y * z).$$

$$1 * x = x.$$

$$x * 1 = x.$$

$$x' * x = 0.$$

$$x * 0 = 0.$$

$$x' * y' = y' * x'.$$

$$x'' * x = x.$$

$$x' = (x * y)' * (x * y')'.$$

$$(x * y)' * x = ((x * y)' * x) * y'.$$

está contida na classe definida pelos seguintes axiomas (Closable semilattice pseudo-complemented semigroup)

$$(x * y) * z = x * (y * z).$$

$$x' * x = 0.$$

$$x * 0' = x.$$

$$x' * y' = y' * x'.$$

$$x' * (x' * y)' = x' * y'.$$

$$x'' * x = x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(1, x) = x.
binary1(x, 1) = x.
binary1(unary1(x), x) = 0.
binary1(x, 0) = 0.
binary1(unary1(x), unary1(y)) = binary1(unary1(y), unary1(x)).
binary1(unary1(unary1(x)), x) = x.
unary1(x) = binary1((unary1(binary1(x, y))), (unary1(binary1(x, unary1(y))))).
binary1((unary1(binary1(x, y))), x) =
binary1((binary1((unary1(binary1(x, y))), x)), unary1(y)).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(unary1(x), x) = 0.
binary1(x, unary1(0)) = x.
binary1(unary1(x), unary1(y)) = binary1(unary1(y), unary1(x)).
binary1(unary1(x), (unary1(binary1(unary1(x), y)))) =
binary1(unary1(x), unary1(y)).
binary1(unary1(unary1(x)), x) = x.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/123`, definition #123 implies definition #125. Number of proofs: 6. Max length proof: 19. □

Teorema 4.5.62 (Antidomain monoid implica Semilattice pseudo-complemented semigroup)**.** *A classe definida pelos seguintes axiomas (Antidomain monoid)*

$$(x * y) * z = x * (y * z).$$

$$1 * x = x.$$

$$x * 1 = x.$$

$$x' * x = 0.$$

$$x * 0 = 0.$$

$$x' * y' = y' * x'.$$

$$x'' * x = x.$$

$$x' = (x * y)' * (x * y')'.$$

$$(x * y)' * x = ((x * y)' * x) * y'.$$

está contida na classe definida pelos seguintes axiomas (Semilattice pseudo-complemented semigroup)

$$(x * y) * z = x * (y * z).$$

$$x' * x = 0.$$

$$x * 0' = x.$$

$$x' * y' = y' * x'.$$

$$x' * (x' * y)' = x' * y'.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(1, x) = x.
binary1(x, 1) = x.
binary1(unary1(x), x) = 0.
binary1(x, 0) = 0.
binary1(unary1(x), unary1(y)) = binary1(unary1(y), unary1(x)).
binary1(unary1(unary1(x)), x) = x.
unary1(x) = binary1((unary1(binary1(x, y))), (unary1(binary1(x, unary1(y))))).
binary1((unary1(binary1(x, y))), x) =
binary1((binary1((unary1(binary1(x, y))), x)), unary1(y)).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(unary1(x), x) = 0.
binary1(x, unary1(0)) = x.
binary1(unary1(x), unary1(y)) = binary1(unary1(y), unary1(x)).
binary1(unary1(x), (unary1(binary1(unary1(x), y)))) =
binary1(unary1(x), unary1(y)).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/123`, definition #123 implies definition #124. Number of proofs: 5. Max length proof: 19. □

Teorema 4.5.63 (Commutative regular rings implica Semirings with identity and zero)**.** *A classe definida pelos seguintes axiomas (Commutative regular rings)*

$$(x + y) + z = x + (y + z).$$

$$x + y = y + x.$$

$$x + 0 = x.$$

$$x + \neg x = 0.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$\exists\, y (x * y) * x = x.$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Semirings with identity and zero)

$$(x * y) * z = x * (y * z).$$

$$x + y = y + x.$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$x * 0 = 0.$$

$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
(exists y binary2((binary2(x, y)), x) = x).
binary2(x, y) = binary2(y, x).
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/27`, definition #27 implies definition #96. Number of proofs: 8. Max length proof: 18. □

Teorema 4.5.64 (Commutative regular rings implica Semirings with zero)**.** *A classe definida pelos seguintes axiomas (Commutative regular rings)*

$$(x + y) + z = x + (y + z).$$

$$x + y = y + x.$$

$$x + 0 = x.$$

$$x + \neg x = 0.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$\exists\, y (x * y) * x = x.$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Semirings with zero)

$$(x * y) * z = x * (y * z).$$

$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
(exists y binary2((binary2(x, y)), x) = x).
binary2(x, y) = binary2(y, x).
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/27`, definition #27 implies definition #97. Number of proofs: 6. Max length proof: 18. □

Teorema 4.5.65 (Commutative regular rings implica Shells)**.** *A classe definida pelos seguintes axiomas (Commutative regular rings)*

$$(x + y) + z = x + (y + z).$$
$$x + y = y + x.$$
$$x + 0 = x.$$
$$x + \neg x = 0.$$
$$(x * y) * z = x * (y * z).$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$\exists\, y (x * y) * x = x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$

$$x + 0 = x.$$

$$1 * x = x.$$

$$x * 1 = x.$$

$$0 * x = 0.$$

$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
(exists y binary2((binary2(x, y)), x) = x).
binary2(x, y) = binary2(y, x).
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/27`, definition #27 implies definition #99. Number of proofs: 6. Max length proof: 18. □

Teorema 4.5.66 (Commutative regular rings implica Semigroups with zero)**.** *A classe definida pelos seguintes axiomas (Commutative regular rings)*

$$(x + y) + z = x + (y + z).$$

$$x + y = y + x.$$

$$x + 0 = x.$$

$$x + \neg x = 0.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$\exists\, y (x * y) * x = x.$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Semigroups with zero)

$$(x * y) * z = x * (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2(x, 0) = x.
binary2(x, unary1(x)) = 0.
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, (binary2(y, z))) = binary2((binary1(x, y)), (binary1(x, z))).
binary1((binary2(x, y)), z) = binary2((binary1(x, z)), (binary1(y, z))).
binary1(x, 1) = x.
binary1(1, x) = x.
(exists y binary1((binary1(x, y)), x) = x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, 0) = 0.
binary1(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/27`, definition #27 implies definition #94. Number of proofs: 3. Max length proof: 18. □

Teorema 4.5.67 (Ockham algebras implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #100. Number of proofs: 6. Max length proof: 18. □

Teorema 4.5.68 (Boolean algebra implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee x = x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge x = x.$$
$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #100. Number of proofs: 6. Max length proof: 17. □

Teorema 4.5.69 (Boolean monoid implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \ \ x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \ \ y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #100. Number of proofs: 6. Max length proof: 17. □

Teorema 4.5.70 (Complemented distributive lattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee x = x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge x = x.$$
$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/29, definition #29 implies definition #100. Number of proofs: 6. Max length proof: 17. □

Teorema 4.5.71 (Commutative dimonoid implica Associative dialgebra)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Associative dialgebra)

$$(x + y) + z = x + (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x * y) + z = x * (y + z).$$
$$(x + y) * z = x * (y * z).$$
$$(x * y) * z = x * (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary1((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary2((binary1(x, y)), z) = binary2(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/153, definition #153 implies definition #157. Number of proofs: 5. Max length proof: 17. □

Teorema 4.5.72 (Boolean algebra implica Complemented modular lattices)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Complemented modular lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$
$$((x \wedge z) \vee y) \wedge z = (x \wedge z) \vee (y \wedge z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
binary2((binary1((binary2(x, z)), y)), z) =
binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition

#31. Number of proofs: 10. Max length proof: 16. □

Teorema 4.5.73 (Boolean monoid implica Complemented modular lattices)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$
$$(x * y) * z = x * (y * z).$$
$$e * x = x.$$
$$x * e = x.$$
$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$
$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Complemented modular lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \ \ x \wedge y = 0).$$
$$((x \wedge z) \vee y) \wedge z = (x \wedge z) \vee (y \wedge z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
binary2((binary1((binary2(x, z)), y)), z) =
binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #31. Number of proofs: 10. Max length proof: 16. □

Teorema 4.5.74 (Complemented distributive lattices implica Complemented modular lattices)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Complemented modular lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$
$$((x \wedge z) \vee y) \wedge z = (x \wedge z) \vee (y \wedge z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
binary2((binary1((binary2(x, z)), y)), z) =
binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #31. Number of proofs: 10. Max length proof: 16. □

Teorema 4.5.75 (Ockham algebras implica Multiplicative semilattices)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Multiplicative semilattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$x \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #66. Number of proofs: 5. Max length proof: 16. □

Teorema 4.5.76 (Ockham algebras implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #71. Number of proofs: 3. Max length proof: 16. □

Teorema 4.5.77 (Commutative dimonoid implica Disemigroup)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$

$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Disemigroup)

$$(x + y) + z = x + (y + z).$$
$$(x * y) * z = x * (y * z).$$
$$x + (y + z) = (x * y) + z.$$
$$x + (y * z) = (x + y) * z.$$
$$x + (y + z) = x + (y * z).$$
$$(x * y) + z = x * (y + z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, (binary1(y, z))) = binary1((binary2(x, y)), z).
binary1(x, (binary2(y, z))) = binary2((binary1(x, y)), z).
binary1(x, (binary1(y, z))) = binary1(x, (binary2(y, z))).
binary1((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/153`, definition #153 implies definition #148. Number of proofs: 6. Max length proof: 16. □

Teorema 4.5.78 (Commutative dimonoid implica Strong doppelsemigroup)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Strong doppelsemigroup)

$$(x + y) * z = x + (y * z).$$
$$(x * y) + z = x * (y + z).$$

$$(x + y) + z = x + (y + z).$$
$$(x * y) * z = x * (y * z).$$
$$x + (y * z) = x * (y + z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary1((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, (binary1(y, z))) = binary1(x, (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/153`, definition #153 implies definition #151. Number of proofs: 5. Max length proof: 16. □

Teorema 4.5.79 (Commutative dimonoid implica Left disemigroup)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Left disemigroup)

$$(x + y) + z = x + (y + z).$$
$$(x * y) * z = x * (y * z).$$
$$x + (y + z) = (x * y) + z.$$
$$x + (y * z) = (x + y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, (binary1(y, z))) = binary1((binary2(x, y)), z).
binary1(x, (binary2(y, z))) = binary2((binary1(x, y)), z).
```

```
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/153, definition #153 implies definition #146. Number of proofs: 4. Max length proof: 16. □

Teorema 4.5.80 (Commutative dimonoid implica Right disemigroup)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Right disemigroup)

$$(x + y) + z = x + (y + z).$$
$$(x * y) * z = x * (y * z).$$
$$x + (y + z) = x + (y * z).$$
$$(x * y) + z = x * (y + z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, (binary1(y, z))) = binary1(x, (binary2(y, z))).
binary1((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/153, definition #153 implies definition #147. Number of proofs: 4. Max length proof: 16. □

Teorema 4.5.81 (Commutative dimonoid implica Doppelsemigroup)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Doppelsemigroup)

$$(x + y) * z = x + (y * z).$$
$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y + z).$$
$$(x * y) * z = x * (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary1((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/153`, definition #153 implies definition #150. Number of proofs: 4. Max length proof: 16. □

Teorema 4.5.82 (Commutative dimonoid implica Diassociative semigroup)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Diassociative semigroup)

$$(x + y) + z = x + (y + z) \quad \& \quad x + (y + z) = x + (y * z).$$
$$(x * y) + z = x * (y + z).$$
$$(x * y) * z = (x + y) * z \quad \& \; (x + y) * z = x * (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) =
binary2(x, (binary2(y, z))) &
```

```
binary2(x, (binary2(y, z))) =
binary2(x, (binary1(y, z))).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary1((binary1(x, y)), z) =
binary1((binary2(x, y)), z) &
binary1((binary2(x, y)), z) =
binary1(x, (binary1(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/153`, definition #153 implies definition #154. Number of proofs: 3. Max length proof: 16. □

Teorema 4.5.83 (Commutative dimonoid implica Duplicial semigroup)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Duplicial semigroup)

$$(x + y) + z = x + (y + z).$$
$$(x * y) + z = x * (y + z).$$
$$(x * y) * z = x * (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/153`, definition #153 implies definition #155. Number of proofs: 3. Max length proof: 16. □

Teorema 4.5.84 (Commutative dimonoid implica Associative dimonoid)**.** *A classe definida pelos seguintes axiomas (Commutative dimonoid)*

$$(x * y) + z = x * (y + z).$$
$$(x + y) + z = x + (y * z).$$
$$(x + y) * z = x * (y * z).$$
$$x + y = y + x.$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Associative dimonoid)

$$x + (y + z) = (x + y) + z \quad \& \quad (x + y) + z = x + (y * z).$$
$$(x * y) + z = x * (y + z).$$
$$(x + y) * z = x * (y * z) \quad \& \; x * (y * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary1(y, z))).
binary1((binary2(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
end_of_list.
formulas(goals).
binary2(x, (binary2(y, z))) =
binary2((binary2(x, y)), z) &
binary2((binary2(x, y)), z) =
binary2(x, (binary1(y, z))).
binary2((binary1(x, y)), z) = binary1(x, (binary2(y, z))).
binary1((binary2(x, y)), z) =
binary1(x, (binary1(y, z))) &
binary1(x, (binary1(y, z))) =
binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/153`, definition #153 implies definition #160. Number of proofs: 3. Max length proof: 16. □

Teorema 4.5.85 (Bilattice implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Bilattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$(x + y) + z = x + (y + z).$$
$$x + y = y + x.$$
$$(x * y) * z = x * (y * z).$$
$$x * y = y * x.$$
$$(x + y) * x = x.$$
$$(x * y) + x = x.$$
$$\neg(x \vee y) = \neg x \wedge \neg y.$$

$$\neg\neg x = x.$$
$$\neg(x + y) = \neg x + \neg y.$$
$$\neg(x * y) = \neg x * \neg y.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, y) = binary3(y, x).
binary4((binary4(x, y)), z) = binary4(x, (binary4(y, z))).
binary4(x, y) = binary4(y, x).
binary4((binary3(x, y)), x) = x.
binary3((binary4(x, y)), x) = x.
unary1(binary1(x, y)) = binary2(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
unary1(binary3(x, y)) = binary3(unary1(x), unary1(y)).
unary1(binary4(x, y)) = binary4(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/7`, definition #7 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.86 (Bilattice implica Directoids)**.** *A classe definida pelos seguintes axiomas (Bilattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$

$$(x + y) + z = x + (y + z).$$
$$x + y = y + x.$$
$$(x * y) * z = x * (y * z).$$
$$x * y = y * x.$$
$$(x + y) * x = x.$$
$$(x * y) + x = x.$$
$$\neg(x \vee y) = \neg x \wedge \neg y.$$
$$\neg\neg x = x.$$
$$\neg(x + y) = \neg x + \neg y.$$
$$\neg(x * y) = \neg x * \neg y.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, y) = binary3(y, x).
binary4((binary4(x, y)), z) = binary4(x, (binary4(y, z))).
binary4(x, y) = binary4(y, x).
binary4((binary3(x, y)), x) = x.
binary3((binary4(x, y)), x) = x.
unary1(binary1(x, y)) = binary2(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
unary1(binary3(x, y)) = binary3(unary1(x), unary1(y)).
unary1(binary4(x, y)) = binary4(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/7`, definition #7 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.87 (Boolean algebra implica Multiplicative semilattices)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Multiplicative semilattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$x \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #66. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.88 (Boolean algebra implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #71. Number of proofs: 3. Max length proof: 15. □

Teorema 4.5.89 (Boolean monoid implica Multiplicative semilattices)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Multiplicative semilattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$x \vee x = x.$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
```

```
binary1(x, y) = binary1(y, x).
binary1(x, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #66. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.90 (Boolean monoid implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$

$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$

$$x * x = x.$$

$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
```

```
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #71. Number of proofs: 3. Max length proof: 15. □

Teorema 4.5.91 (Bounded lattice implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Bounded lattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
```

```
binary1(0, x) = x.
binary1(1, x) = 1.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/12`, definition #12 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.92 (Bounded lattice implica Directoids)**.** *A classe definida pelos seguintes axiomas (Bounded lattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/12, definition #12 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.93 (Commutative lattice-ordered monoids implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Commutative lattice-ordered monoids)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$(x * y) * z = x * (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, 1) = x.
binary3(1, x) = x.
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
binary3(x, y) = binary3(y, x).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
```

```
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/20, definition #20 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.94 (Commutative lattice-ordered monoids implica Directoids)**.** *A classe definida pelos seguintes axiomas (Commutative lattice-ordered monoids)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$(x * y) * z = x * (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, 1) = x.
binary3(1, x) = x.
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
binary3(x, y) = binary3(y, x).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
```

```
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/20, definition #20 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.95 (Commutative lattice-ordered semigroups implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Commutative lattice-ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$

$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
binary3(x, y) = binary3(y, x).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/22, definition #22 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.96 (Commutative lattice-ordered semigroups implica Directoids)**.** *A classe definida pelos seguintes axiomas (Commutative lattice-ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$(x * y) * z = x * (y * z).$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
binary3(x, y) = binary3(y, x).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/22, definition #22 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.97 (Complemented distributive lattices implica Multiplicative semilattices)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Multiplicative semilattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$x \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #66. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.98 (Complemented distributive lattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #71. Number of proofs: 3. Max length proof: 15. □

Teorema 4.5.99 (Complemented lattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Complemented lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$

$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/30`, definition #30 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.100 (Complemented lattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Complemented lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$

$$(x * y) * x = x * y.$$

$$y * (x * y) = x * y.$$

$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/30`, definition #30 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.101 (Complemented modular lattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Complemented modular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$

$$((x \wedge z) \vee y) \wedge z = (x \wedge z) \vee (y \wedge z).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
binary2((binary1((binary2(x, z)), y)), z) =
binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/31`, definition #31 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.102 (Complemented modular lattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Complemented modular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$
$$((x \wedge z) \vee y) \wedge z = (x \wedge z) \vee (y \wedge z).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
binary2((binary1((binary2(x, z)), y)), z) =
binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/31`, definition #31 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.103 (Distributive lattice ordered semigroups implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Distributive lattice ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x * y) * z = x * (y * z).$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge x = x.$$
$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/36`, definition #36 implies definition #100. Number of proofs: 6. Max length proof: 15. □

Teorema 4.5.104 (Implicative lattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Implicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \circ (y \vee z) = (x \circ y) \vee (x \circ z).$$
$$x \circ (y \wedge z) = (x \circ y) \wedge (x \circ z).$$
$$(x \vee y) \circ z = (x \circ z) \wedge (y \circ z).$$
$$(x \wedge y) \circ z = (x \circ z) \vee (y \circ z).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$

$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3(x, (binary2(y, z))) = binary2((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary2((binary3(x, z)), (binary3(y, z))).
binary3((binary2(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/47`, definition #47 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.105 (Implicative lattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Implicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \circ (y \vee z) = (x \circ y) \vee (x \circ z).$$

$$x \circ (y \wedge z) = (x \circ y) \wedge (x \circ z).$$

$$(x \vee y) \circ z = (x \circ z) \wedge (y \circ z).$$

$$(x \wedge y) \circ z = (x \circ z) \vee (y \circ z).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3(x, (binary2(y, z))) = binary2((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary2((binary3(x, z)), (binary3(y, z))).
binary3((binary2(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/47`, definition #47 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.106 (Involutive lattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Involutive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$\neg(x \wedge y) = \neg x \vee \neg y.$$
$$\neg\neg x = x.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
unary1(binary2(x, y)) = binary1(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/50`, definition #50 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.107 (Involutive lattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Involutive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$\neg(x \wedge y) = \neg x \vee \neg y.$$
$$\neg\neg x = x.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
```

```
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
unary1(binary2(x, y)) = binary1(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/50`, definition #50 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.108 (Lattice-ordered semigroups implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Lattice-ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$

$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
```

```
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/54`, definition #54 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.109 (Lattice-ordered semigroups implica Directoids)**.** *A classe definida pelos seguintes axiomas (Lattice-ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$

$$(x * y) * x = x * y.$$

$$y * (x * y) = x * y.$$

$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/54, definition #54 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.110 (Multiplicative lattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Multiplicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/65, definition #65 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.111 (Multiplicative lattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Multiplicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/65`, definition #65 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.112 (Near-ring implica Clifford semigroup)**.** *A classe definida pelos seguintes axiomas (Near-ring)*

$$(x + y) + z = x + (y + z).$$
$$x + 0 = x.$$
$$x + \neg x = 0.$$
$$(x * y) * z = x * (y * z).$$
$$(x + y) * z = (x * z) + (y * z).$$

está contida na classe definida pelos seguintes axiomas (Clifford semigroup)

$$(x * y) * z = x * (y * z).$$
$$x'' = x.$$
$$(x * x') * x = x.$$
$$(x * x') * (y * y') = (y * y') * (x * x').$$
$$(x * x') * x = x.$$
$$(x' * x) * x' = x'.$$
$$x * x' = x' * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1(x, unary1(x))), (binary1(y, unary1(y)))) =
binary1((binary1(y, unary1(y))), (binary1(x, unary1(x)))).
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1(unary1(x), x)), unary1(x)) = unary1(x).
binary1(x, unary1(x)) = binary1(unary1(x), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/68`, definition #68 implies definition #17. Number of proofs: 7. Max length proof: 15. □

Teorema 4.5.113 (Near-rings with identity implica Clifford semigroup)**.** *A classe definida pelos seguintes axiomas (Near-rings with identity)*

$$(x + y) + z = x + (y + z).$$
$$x + 0 = x.$$
$$x + \neg x = 0.$$
$$(x * y) * z = x * (y * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$

está contida na classe definida pelos seguintes axiomas (Clifford semigroup)

$$(x * y) * z = x * (y * z).$$
$$x'' = x.$$
$$(x * x') * x = x.$$

$$(x * x') * (y * y') = (y * y') * (x * x').$$
$$(x * x') * x = x.$$
$$(x' * x) * x' = x'.$$
$$x * x' = x' * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1(x, unary1(x))), (binary1(y, unary1(y)))) =
binary1((binary1(y, unary1(y))), (binary1(x, unary1(x)))).
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1(unary1(x), x)), unary1(x)) = unary1(x).
binary1(x, unary1(x)) = binary1(unary1(x), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/69`, definition #69 implies definition #17. Number of proofs: 7. Max length proof: 15. □

Teorema 4.5.114 (Neardistributive lattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Neardistributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y)))))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y)))))))))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/70`, definition #70 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.115 (Neardistributive lattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Neardistributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
```

```
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y)))))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y)))))))))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/70`, definition #70 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.116 (Ortholattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Ortholattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
```

```
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/80`, definition #80 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.117 (Ortholattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Ortholattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
```

```
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/80`, definition #80 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.118 (Orthomodular lattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Orthomodular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$
$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
binary1(x, (binary2(unary1(x), (binary1(x, y))))) = binary1(x, y).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/81`, definition #81 implies definition #74. Number of proofs: 5. Max length proof: 15. □

Teorema 4.5.119 (Orthomodular lattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Orthomodular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$
$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$

$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
binary1(x, (binary2(unary1(x), (binary1(x, y))))) = binary1(x, y).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/81`, definition #81 implies definition #35. Number of proofs: 4. Max length proof: 15. □

Teorema 4.5.120 (Distributive lattice ordered semigroups implica Multiplicative semilattices)**.** *A classe definida pelos seguintes axiomas (Distributive lattice ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x * y) * z = x * (y * z).$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Multiplicative semilattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$x \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/36`, definition #36 implies definition #66. Number of proofs: 5. Max length proof: 14. □

Teorema 4.5.121 (Distributive lattice ordered semigroups implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Distributive lattice ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$(x * y) * z = x * (y * z).$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$

$$x * x = x.$$

$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
```

```
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/36`, definition #36 implies definition #71. Number of proofs: 3. Max length proof: 14. □

Teorema 4.5.122 (Bilattice implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Bilattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x + y) + z = x + (y + z).$$

$$x + y = y + x.$$

$$(x * y) * z = x * (y * z).$$

$$x * y = y * x.$$

$$(x + y) * x = x.$$

$$(x * y) + x = x.$$

$$\neg(x \vee y) = \neg x \wedge \neg y.$$

$$\neg\neg x = x.$$

$$\neg(x + y) = \neg x + \neg y.$$

$$\neg(x * y) = \neg x * \neg y.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, y) = binary3(y, x).
binary4((binary4(x, y)), z) = binary4(x, (binary4(y, z))).
binary4(x, y) = binary4(y, x).
binary4((binary3(x, y)), x) = x.
binary3((binary4(x, y)), x) = x.
unary1(binary1(x, y)) = binary2(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
unary1(binary3(x, y)) = binary3(unary1(x), unary1(y)).
unary1(binary4(x, y)) = binary4(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/7`, definition #7 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.123 (Bounded lattice implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Bounded lattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee x = x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/12`, definition #12 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.124 (Commutative lattice-ordered monoids implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Commutative lattice-ordered monoids)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x * y) * z = x * (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, 1) = x.
binary3(1, x) = x.
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
binary3(x, y) = binary3(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/20`, definition #20 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.125 (Commutative lattice-ordered semigroups implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Commutative lattice-ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
binary3(x, y) = binary3(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/22`, definition #22 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.126 (Complemented lattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Complemented lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge x = x.$$
$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/30`, definition #30 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.127 (Complemented modular lattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Complemented modular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$
$$((x \wedge z) \vee y) \wedge z = (x \wedge z) \vee (y \wedge z).$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
binary2((binary1((binary2(x, z)), y)), z) =
binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/31`, definition #31 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.128 (Implicative lattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Implicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \circ (y \vee z) = (x \circ y) \vee (x \circ z).$$

$$x \circ (y \wedge z) = (x \circ y) \wedge (x \circ z).$$

$$(x \vee y) \circ z = (x \circ z) \wedge (y \circ z).$$

$$(x \wedge y) \circ z = (x \circ z) \vee (y \circ z).$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3(x, (binary2(y, z))) = binary2((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary2((binary3(x, z)), (binary3(y, z))).
binary3((binary2(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/47`, definition #47 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.129 (Involutive lattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Involutive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$\neg(x \wedge y) = \neg x \vee \neg y.$$

$$\neg\neg x = x.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
unary1(binary2(x, y)) = binary1(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/50`, definition #50 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.130 (Lattice-ordered semigroups implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Lattice-ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/54`, definition #54 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.131 (Multiplicative lattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Multiplicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee x = x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge x = x.$$
$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/65`, definition #65 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.132 (Near-ring implica Inverse semigroup)**.** *A classe definida pelos seguintes axiomas (Near-ring)*

$$(x + y) + z = x + (y + z).$$

$$x + 0 = x.$$

$$x + \neg x = 0.$$

$$(x * y) * z = x * (y * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

está contida na classe definida pelos seguintes axiomas (Inverse semigroup)

$$(x * y) * z = x * (y * z).$$

$$x'' = x.$$

$$(x * x') * x = x.$$

$$((x * x') * y) * y' = ((y * y') * x) * x'.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1((binary1(x, unary1(x))), y)), unary1(y)) =
binary1((binary1((binary1(y, unary1(y))), x)), unary1(x)).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/68, definition #68 implies definition #121. Number of proofs: 4. Max length proof: 13. □

Teorema 4.5.133 (Near-rings with identity implica Inverse semigroup)**.** *A classe definida pelos seguintes axiomas (Near-rings with identity)*

$$(x + y) + z = x + (y + z).$$
$$x + 0 = x.$$
$$x + \neg x = 0.$$
$$(x * y) * z = x * (y * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$

está contida na classe definida pelos seguintes axiomas (Inverse semigroup)

$$(x * y) * z = x * (y * z).$$
$$x'' = x.$$
$$(x * x') * x = x.$$
$$((x * x') * y) * y' = ((y * y') * x) * x'.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1((binary1(x, unary1(x))), y)), unary1(y)) =
binary1((binary1((binary1(y, unary1(y))), x)), unary1(x)).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/69, definition #69 implies definition #121. Number of proofs: 4. Max length proof: 13. □

Teorema 4.5.134 (Neardistributive lattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Neardistributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee x = x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge x = x.$$
$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y)))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y)))))))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/70`, definition #70 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.135 (Ortholattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Ortholattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee x = x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge x = x.$$
$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$
$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/80`, definition #80 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.136 (Orthomodular lattices implica Skew lattices)**.** *A classe definida pelos seguintes axiomas (Orthomodular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \vee x' = 1.$$

$$x \wedge x' = 0.$$

$$(x \vee y)' = x' \wedge y'.$$

$$(x \wedge y)' = x' \vee y'.$$

$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

está contida na classe definida pelos seguintes axiomas (Skew lattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee x = x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge x = x.$$

$$x \wedge (x \vee y) = x \quad \& \quad x = x \vee (x \wedge y).$$

$$(x \vee y) \wedge y = y \quad \& \quad y = (x \wedge y) \vee y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
binary1(x, (binary2(unary1(x), (binary1(x, y))))) = binary1(x, y).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, x) = x.
binary2(x, (binary1(x, y))) = x & x = binary1(x, (binary2(x, y))).
binary2((binary1(x, y)), y) = y & y = binary1((binary2(x, y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/81`, definition #81 implies definition #100. Number of proofs: 6. Max length proof: 13. □

Teorema 4.5.137 (Extra loop implica Right conjugacy closed loop)**.** *A classe definida pelos seguintes axiomas (Extra loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * (y * z)) * y = (x * y) * (z * y).$$

está contida na classe definida pelos seguintes axiomas (Right conjugacy closed loop)

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * y) * z = (x * z) * (z \setminus (y * z)).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, (binary1(y, z)))), y) =
binary1((binary1(x, y)), (binary1(z, y))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, y)), z) =
binary1((binary1(x, z)), (binary2(z, (binary1(y, z))))).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/135`, definition #135 implies definition #131. Number of proofs: 7. Max length proof: 13. □

Teorema 4.5.138 (Bounded lattice implica Shells)**.** *A classe definida pelos seguintes axiomas (Bounded lattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$
$$x + 0 = x.$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$0 * x = 0.$$
$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/12`, definition #12 implies definition #99. Number of proofs: 6. Max length proof: 12. □

Teorema 4.5.139 (Loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (Loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$

$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * e = x.$$
$$x = e * x.$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, nullary1) = x.
x = binary1(nullary1, x).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/32`, definition #32 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.140 (Left Bol loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (Left Bol loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$

$$x = 1 * x.$$
$$x * (y * (x * z)) = (x * (y * x)) * z.$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1(x, (binary1(y, (binary1(x, z))))) =
binary1((binary1(x, (binary1(y, x)))), z).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/128`, definition #128 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.141 (Right Bol loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (Right Bol loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$

$$z * ((x * y) * x) = ((z * x) * y) * x.$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1(z, (binary1((binary1(x, y)), x))) =
binary1((binary1((binary1(z, x)), y)), x).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/129`, definition #129 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.142 (Left conjugacy closed loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (Left conjugacy closed loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$z * (y * x) = ((z * y) \,/\, z) * (z * x).$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1(z, (binary1(y, x))) =
binary1((binary3((binary1(z, y)), z)), (binary1(z, x))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/130`, definition #130 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.143 (Right conjugacy closed loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (Right conjugacy closed loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * y) * z = (x * z) * (z \setminus (y * z)).$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, y)), z) =
binary1((binary1(x, z)), (binary2(z, (binary1(y, z))))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/131`, definition #131 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.144 (Conjugacy closed loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (Conjugacy closed loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$z * (y * x) = ((z * y) \,/\, z) * (z * x).$$
$$(x * y) * z = (x * z) * (z \setminus (y * z)).$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1(z, (binary1(y, x))) =
binary1((binary3((binary1(z, y)), z)), (binary1(z, x))).
binary1((binary1(x, y)), z) =
binary1((binary1(x, z)), (binary2(z, (binary1(y, z))))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/132`, definition #132 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.145 (RIF loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (RIF loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * y) * (z * (x * y)) = ((x * (y * z)) * x) * y.$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, y)), (binary1(z, (binary1(x, y))))) =
binary1((binary1((binary1(x, (binary1(y, z)))), x)), y).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/133`, definition #133 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.146 (ARIF loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (ARIF loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(z * x) * ((y * x) * y) = (z * ((x * y) * x)) * y.$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(z, x)), (binary1((binary1(y, x)), y))) =
binary1((binary1(z, (binary1((binary1(x, y)), x)))), y).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/134`, definition #134 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.147 (Extra loop implica Rectangular loop)**.** *A classe definida pelos seguintes axiomas (Extra loop)*

$$x \setminus (x * y) = y.$$
$$x * (x \setminus y) = y.$$
$$(x * y) \,/\, y = x.$$
$$(x \,/\, y) * y = x.$$
$$x * 1 = x.$$
$$x = 1 * x.$$
$$(x * (y * z)) * y = (x * y) * (z * y).$$

está contida na classe definida pelos seguintes axiomas (Rectangular loop)

$$x \setminus (x * x) = x.$$
$$(x * x) \,/\, x = x.$$
$$x * (x \setminus y) = x \setminus (x * y).$$
$$(x \,/\, y) * y = (x * y) \,/\, y.$$
$$(x \setminus y) \setminus ((x \setminus y) * (z * u)) = (x \setminus (x * z)) * u.$$
$$((x * y) * (z \,/\, u)) \,/\, (z \,/\, u) = x * ((y * u) \,/\, u).$$
$$x \setminus (x * (y \setminus y)) = ((x \,/\, x) * y) \,/\, y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2(x, (binary1(x, y))) = y.
binary1(x, (binary2(x, y))) = y.
binary3((binary1(x, y)), y) = x.
binary1((binary3(x, y)), y) = x.
binary1(x, 1) = x.
x = binary1(1, x).
binary1((binary1(x, (binary1(y, z)))), y) =
binary1((binary1(x, y)), (binary1(z, y))).
end_of_list.
formulas(goals).
binary2(x, (binary1(x, x))) = x.
binary3((binary1(x, x)), x) = x.
binary1(x, (binary2(x, y))) = binary2(x, (binary1(x, y))).
binary1((binary3(x, y)), y) = binary3((binary1(x, y)), y).
binary2((binary2(x, y)), (binary1((binary2(x, y)), (binary1(z, u))))) =
binary1((binary2(x, (binary1(x, z)))), u).
binary3((binary1((binary1(x, y)), (binary3(z, u)))), (binary3(z, u))) =
binary1(x, (binary3((binary1(y, u)), u))).
binary2(x, (binary1(x, (binary2(y, y))))) =
binary3((binary1((binary3(x, x)), y)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/135`, definition #135 implies definition #140. Number of proofs: 7. Max length proof: 12. □

Teorema 4.5.148 (Brouwerian semilattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Brouwerian semilattices)*

$$x * y = y * x.$$
$$(x * y) * z = x * (y * z).$$
$$1 * x = x.$$
$$x \circ (y \circ z) = (x * y) \circ z.$$
$$x \circ x = 1.$$
$$(x \circ y) * x = (y \circ x) * y.$$
$$x * x = x.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1(x, y) = binary1(y, x).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(1, x) = x.
binary2(x, (binary2(y, z))) = binary2((binary1(x, y)), z).
binary2(x, x) = 1.
binary1((binary2(x, y)), x) = binary1((binary2(y, x)), y).
binary1(x, x) = x.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/15`, definition #15 implies definition #74. Number of proofs: 5. Max length proof: 11. □

Teorema 4.5.149 (Brouwerian semilattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Brouwerian semilattices)*

$$x * y = y * x.$$
$$(x * y) * z = x * (y * z).$$
$$1 * x = x.$$
$$x \circ (y \circ z) = (x * y) \circ z.$$
$$x \circ x = 1.$$
$$(x \circ y) * x = (y \circ x) * y.$$
$$x * x = x.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1(x, y) = binary1(y, x).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(1, x) = x.
binary2(x, (binary2(y, z))) = binary2((binary1(x, y)), z).
binary2(x, x) = 1.
binary1((binary2(x, y)), x) = binary1((binary2(y, x)), y).
binary1(x, x) = x.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/15`, definition #15 implies definition #35. Number of proofs: 4. Max length proof: 11. □

Teorema 4.5.150 (Complemented lattices implica Shells)**.** *A classe definida pelos seguintes axiomas (Complemented lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$
$$x + 0 = x.$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$0 * x = 0.$$
$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
```

```
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/30`, definition #30 implies definition #99. Number of proofs: 6. Max length proof: 11. □

Teorema 4.5.151 (Complemented modular lattices implica Shells)**.** *A classe definida pelos seguintes axiomas (Complemented modular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$
$$((x \wedge z) \vee y) \wedge z = (x \wedge z) \vee (y \wedge z).$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$
$$x + 0 = x.$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$0 * x = 0.$$
$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
```

```
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
binary2((binary1((binary2(x, z)), y)), z) =
binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/31`, definition #31 implies definition #99. Number of proofs: 6. Max length proof: 11. □

Teorema 4.5.152 (Idempotent semiring implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Idempotent semiring)*

$$x + (y + z) = (x + y) + z.$$
$$x + y = y + x.$$
$$x + 0 = x.$$
$$0 + x = x.$$
$$x + x = x.$$
$$x * (y * z) = (x * y) * z.$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$
$$(x * y) * x = y * x.$$
$$(x * y) * y = x * y.$$
$$x * ((x * y) * z) = x * (y * z).$$
$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1(x, (binary1(y, z))) = binary1((binary1(x, y)), z).
```

```
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(0, x) = x.
binary1(x, x) = x.
binary2(x, (binary2(y, z))) = binary2((binary2(x, y)), z).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/110`, definition #110 implies definition #74. Number of proofs: 5. Max length proof: 11. □

Teorema 4.5.153 (Idempotent semiring implica Directoids)**.** *A classe definida pelos seguintes axiomas (Idempotent semiring)*

$$x + (y + z) = (x + y) + z.$$
$$x + y = y + x.$$
$$x + 0 = x.$$
$$0 + x = x.$$
$$x + x = x.$$
$$x * (y * z) = (x * y) * z.$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1(x, (binary1(y, z))) = binary1((binary1(x, y)), z).
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(0, x) = x.
binary1(x, x) = x.
binary2(x, (binary2(y, z))) = binary2((binary2(x, y)), z).
binary2(x, 1) = x.
binary2(1, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/110`, definition #110 implies definition #35. Number of proofs: 4. Max length proof: 11. □

Teorema 4.5.154 (Bilattice implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Bilattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$(x + y) + z = x + (y + z).$$
$$x + y = y + x.$$
$$(x * y) * z = x * (y * z).$$
$$x * y = y * x.$$
$$(x + y) * x = x.$$
$$(x * y) + x = x.$$
$$\neg(x \vee y) = \neg x \wedge \neg y.$$
$$\neg\neg x = x.$$
$$\neg(x + y) = \neg x + \neg y.$$
$$\neg(x * y) = \neg x * \neg y.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$

$$x * x = x.$$

$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, y) = binary3(y, x).
binary4((binary4(x, y)), z) = binary4(x, (binary4(y, z))).
binary4(x, y) = binary4(y, x).
binary4((binary3(x, y)), x) = x.
binary3((binary4(x, y)), x) = x.
unary1(binary1(x, y)) = binary2(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
unary1(binary3(x, y)) = binary3(unary1(x), unary1(y)).
unary1(binary4(x, y)) = binary4(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/7`, definition #7 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.155 (Boolean algebra implica Semirings with zero)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Semirings with zero)

$$(x * y) * z = x * (y * z).$$

$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #97. Number of proofs: 6. Max length proof: 10. □

Teorema 4.5.156 (Boolean algebra implica Semigroups with zero)**.** *A classe definida pelos seguintes axiomas (Boolean algebra)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \wedge \neg x = 0.$$
$$x \vee \neg x = 1.$$

está contida na classe definida pelos seguintes axiomas (Semigroups with zero)

$$(x * y) * z = x * (y * z).$$

$$x * 0 = 0.$$

$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1((binary2(x, y)), x) = x.
binary2((binary1(x, y)), x) = x.
binary2(0, x) = x.
binary2(1, x) = 1.
binary1(x, (binary2(y, z))) = binary2((binary1(x, y)), (binary1(x, z))).
binary1(x, unary1(x)) = 0.
binary2(x, unary1(x)) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, 0) = 0.
binary1(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/9`, definition #9 implies definition #94. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.157 (Boolean monoid implica Semirings with zero)**.** *A classe definida pelos seguintes axiomas (Boolean monoid)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

$$x \wedge \neg x = 0.$$

$$x \vee \neg x = 1.$$

$$(x * y) * z = x * (y * z).$$

$$e * x = x.$$

$$x * e = x.$$

$$((x \vee y) * z = (x * z) \vee (y * z)) \quad \& \ (x * (y \vee z) = (x * y) \vee (x * z)).$$
$$(0 * x = 0) \quad \& \ \ (x * 0 = 0).$$

está contida na classe definida pelos seguintes axiomas (Semirings with zero)

$$(x * y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2(x, unary1(x)) = 0.
binary1(x, unary1(x)) = 1.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(nullary1, x) = x.
binary3(x, nullary1) = x.
binary3((binary1(x, y)), z) =
binary1((binary3(x, z)), (binary3(y, z))) &
binary3(x, (binary1(y, z))) =
binary1((binary3(x, y)), (binary3(x, z))).
binary3(0, x) = 0 & binary3(x, 0) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/10`, definition #10 implies definition #97. Number of proofs: 6. Max length proof: 10. □

Teorema 4.5.158 (Bounded lattice implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Bounded lattice)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/12`, definition #12 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.159 (Clifford semigroup implica Inverse semigroup)**.** *A classe definida pelos seguintes axiomas (Clifford semigroup)*

$$(x * y) * z = x * (y * z).$$
$$x'' = x.$$
$$(x * x') * x = x.$$
$$(x * x') * (y * y') = (y * y') * (x * x').$$
$$(x * x') * x = x.$$
$$(x' * x) * x' = x'.$$
$$x * x' = x' * x.$$

está contida na classe definida pelos seguintes axiomas (Inverse semigroup)

$$(x * y) * z = x * (y * z).$$
$$x'' = x.$$

$$(x * x') * x = x.$$
$$((x * x') * y) * y' = ((y * y') * x) * x'.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1(x, unary1(x))), (binary1(y, unary1(y)))) =
binary1((binary1(y, unary1(y))), (binary1(x, unary1(x)))).
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1(unary1(x), x)), unary1(x)) = unary1(x).
binary1(x, unary1(x)) = binary1(unary1(x), x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1((binary1(x, unary1(x))), y)), unary1(y)) =
binary1((binary1((binary1(y, unary1(y))), x)), unary1(x)).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/17`, definition #17 implies definition #121. Number of proofs: 4. Max length proof: 10. □

Teorema 4.5.160 (Commutative lattice-ordered monoids implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Commutative lattice-ordered monoids)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$(x * y) * z = x * (y * z).$$
$$x * 1 = x.$$
$$1 * x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$
$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, 1) = x.
binary3(1, x) = x.
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
binary3(x, y) = binary3(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/20`, definition #20 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.161 (Commutative lattice-ordered semigroups implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Commutative lattice-ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$

$$x * x = x.$$

$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
```

```
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
binary3(x, y) = binary3(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/22`, definition #22 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.162 (Commutative regular rings implica Clifford semigroup)**.** *A classe definida pelos seguintes axiomas (Commutative regular rings)*

$$(x + y) + z = x + (y + z).$$

$$x + y = y + x.$$

$$x + 0 = x.$$

$$x + \neg x = 0.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$\exists\, y (x * y) * x = x.$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Clifford semigroup)

$$(x * y) * z = x * (y * z).$$

$$x'' = x.$$

$$(x * x') * x = x.$$

$$(x * x') * (y * y') = (y * y') * (x * x').$$

$$(x * x') * x = x.$$

$$(x' * x) * x' = x'.$$

$$x * x' = x' * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
```

```
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
(exists y binary2((binary2(x, y)), x) = x).
binary2(x, y) = binary2(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1(x, unary1(x))), (binary1(y, unary1(y)))) =
binary1((binary1(y, unary1(y))), (binary1(x, unary1(x)))).
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1(unary1(x), x)), unary1(x)) = unary1(x).
binary1(x, unary1(x)) = binary1(unary1(x), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/27`, definition #27 implies definition #17. Number of proofs: 7. Max length proof: 10. □

Teorema 4.5.163 (Commutative regular rings implica Inverse semigroup)**.** *A classe definida pelos seguintes axiomas (Commutative regular rings)*

$$(x + y) + z = x + (y + z).$$

$$x + y = y + x.$$

$$x + 0 = x.$$

$$x + \neg x = 0.$$

$$(x * y) * z = x * (y * z).$$

$$x * (y + z) = (x * y) + (x * z).$$

$$(x + y) * z = (x * z) + (y * z).$$

$$x * 1 = x.$$

$$1 * x = x.$$

$$\exists\, y (x * y) * x = x.$$

$$x * y = y * x.$$

está contida na classe definida pelos seguintes axiomas (Inverse semigroup)

$$(x * y) * z = x * (y * z).$$

$$x'' = x.$$

$$(x * x') * x = x.$$

$$((x * x') * y) * y' = ((y * y') * x) * x'.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, 0) = x.
binary1(x, unary1(x)) = 0.
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 1) = x.
binary2(1, x) = x.
(exists y binary2((binary2(x, y)), x) = x).
binary2(x, y) = binary2(y, x).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
unary1(unary1(x)) = x.
binary1((binary1(x, unary1(x))), x) = x.
binary1((binary1((binary1(x, unary1(x))), y)), unary1(y)) =
binary1((binary1((binary1(y, unary1(y))), x)), unary1(x)).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/27, definition #27 implies definition #121. Number of proofs: 4. Max length proof: 10. □

Teorema 4.5.164 (Complemented distributive lattices implica Semirings with zero)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Semirings with zero)

$$(x * y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #97. Number of proofs: 6. Max length proof: 10. □

Teorema 4.5.165 (Complemented distributive lattices implica Semigroups with zero)**.** *A classe definida pelos seguintes axiomas (Complemented distributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$

está contida na classe definida pelos seguintes axiomas (Semigroups with zero)

$$(x * y) * z = x * (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2(x, y) = binary2(y, x).
binary1(x, y) = binary1(y, x).
binary1((binary2(x, y)), x) = x.
binary2((binary1(x, y)), x) = x.
binary2(0, x) = x.
binary2(1, x) = 1.
binary1(x, (binary2(y, z))) = binary2((binary1(x, y)), (binary1(x, z))).
binary2(x, unary1(x)) = 1.
binary1(x, unary1(x)) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, 0) = 0.
binary1(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/29`, definition #29 implies definition #94. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.166 (Complemented lattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Complemented lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
```

```
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/30`, definition #30 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.167 (Complemented modular lattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Complemented modular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$

$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$

$$1 \vee x = 1.$$

$$\forall\, x \exists\, y (x \vee y = 1 \quad \& \quad x \wedge y = 0).$$

$$((x \wedge z) \vee y) \wedge z = (x \wedge z) \vee (y \wedge z).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$

$$x * x = x.$$

$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
(all x exists y (binary1(x, y) = 1 & binary2(x, y) = 0)).
binary2((binary1((binary2(x, z)), y)), z) =
binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/31, definition #31 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.168 (Implicative lattices implica Multiplicative semilattices)**.** *A classe definida pelos seguintes axiomas (Implicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \circ (y \vee z) = (x \circ y) \vee (x \circ z).$$
$$x \circ (y \wedge z) = (x \circ y) \wedge (x \circ z).$$
$$(x \vee y) \circ z = (x \circ z) \wedge (y \circ z).$$
$$(x \wedge y) \circ z = (x \circ z) \vee (y \circ z).$$

está contida na classe definida pelos seguintes axiomas (Multiplicative semilattices)

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$x \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3(x, (binary2(y, z))) = binary2((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary2((binary3(x, z)), (binary3(y, z))).
binary3((binary2(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/47, definition #47 implies definition #66. Number of proofs: 5. Max length proof: 10. □

Teorema 4.5.169 (Implicative lattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Implicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$x \circ (y \vee z) = (x \circ y) \vee (x \circ z).$$
$$x \circ (y \wedge z) = (x \circ y) \wedge (x \circ z).$$
$$(x \vee y) \circ z = (x \circ z) \wedge (y \circ z).$$
$$(x \wedge y) \circ z = (x \circ z) \vee (y \circ z).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3(x, (binary2(y, z))) = binary2((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary2((binary3(x, z)), (binary3(y, z))).
binary3((binary2(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/47, definition #47 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.170 (Involutive lattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Involutive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$\neg(x \wedge y) = \neg x \vee \neg y.$$
$$\neg\neg x = x.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
unary1(binary2(x, y)) = binary1(unary1(x), unary1(y)).
unary1(unary1(x)) = x.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/50`, definition #50 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.171 (Lattice-ordered semigroups implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Lattice-ordered semigroups)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$

$$(x \wedge y) \vee x = x.$$
$$(x * y) * z = x * (y * z).$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3((binary3(x, y)), z) = binary3(x, (binary3(y, z))).
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/54`, definition #54 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.172 (Multiplicative lattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Multiplicative lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$

$$x * x = x.$$

$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary3(x, (binary1(y, z))) = binary1((binary3(x, y)), (binary3(x, z))).
binary3((binary1(x, y)), z) = binary1((binary3(x, z)), (binary3(y, z))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/65`, definition #65 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.173 (Multiplicative semilattices implica Order algebras)**.** *A classe definida pelos seguintes axiomas (Multiplicative semilattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$

$$x \vee y = y \vee x.$$

$$x \vee x = x.$$

$$x * (y \vee z) = (x * y) \vee (x * z).$$

$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Order algebras)

$$x * x = x.$$

$$(x * y) * x = y * x.$$

$$(x * y) * y = x * y.$$

$$x * ((x * y) * z) = x * (y * z).$$

$$((x * y) * z) * y = (x * z) * y.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(y, x).
```

```
binary1((binary1(x, y)), y) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1(x, (binary1(y, z))).
binary1((binary1((binary1(x, y)), z)), y) = binary1((binary1(x, z)), y).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/66, definition #66 implies definition #74. Number of proofs: 5. Max length proof: 10. □

Teorema 4.5.174 (Multiplicative semilattices implica Directoids)**.** *A classe definida pelos seguintes axiomas (Multiplicative semilattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$x \vee x = x.$$
$$x * (y \vee z) = (x * y) \vee (x * z).$$
$$(x \vee y) * z = (x * z) \vee (y * z).$$

está contida na classe definida pelos seguintes axiomas (Directoids)

$$x * x = x.$$
$$(x * y) * x = x * y.$$
$$y * (x * y) = x * y.$$
$$x * ((x * y) * z) = (x * y) * z.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1(x, x) = x.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
end_of_list.
formulas(goals).
binary1(x, x) = x.
binary1((binary1(x, y)), x) = binary1(x, y).
binary1(y, (binary1(x, y))) = binary1(x, y).
binary1(x, (binary1((binary1(x, y)), z))) = binary1((binary1(x, y)), z).
end_of_list.
```

Ou em https://marciedb.pythonanywhere.com/definition/66, definition #66 implies definition #35. Number of proofs: 4. Max length proof: 10. □

Teorema 4.5.175 (Neardistributive lattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Neardistributive lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$x \wedge (y \vee z) = x \wedge (y \vee (x \wedge (z \vee (x \wedge y)))).$$
$$x \vee (y \wedge z) = x \vee (y \wedge (x \vee (z \wedge (x \vee y)))).$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary2(x, (binary1(y, z))) =
binary2(x, (binary1(y, (binary2(x, (binary1(z, (binary2(x, y))))))))).
binary1(x, (binary2(y, z))) =
binary1(x, (binary2(y, (binary1(x, (binary2(z, (binary1(x, y))))))))).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/70`, definition #70 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.176 (Ockham algebras implica Semirings with zero)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$

$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Semirings with zero)

$$(x * y) * z = x * (y * z).$$
$$x + y = y + x.$$
$$x * (y + z) = (x * y) + (x * z).$$
$$(x + y) * z = (x * z) + (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, (binary1(y, z))) = binary1((binary2(x, y)), (binary2(x, z))).
binary2((binary1(x, y)), z) = binary1((binary2(x, z)), (binary2(y, z))).
binary2(x, 0) = 0.
binary2(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #97. Number of proofs: 6. Max length proof: 10. □

Teorema 4.5.177 (Ockham algebras implica Semigroups with zero)**.** *A classe definida pelos seguintes axiomas (Ockham algebras)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$x \vee y = y \vee x.$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$
$$(x \wedge y)' = x' \vee y'.$$
$$(x \vee y)' = x' \wedge y'.$$
$$0' = 1.$$
$$1' = 0.$$

está contida na classe definida pelos seguintes axiomas (Semigroups with zero)

$$(x * y) * z = x * (y * z).$$
$$x * 0 = 0.$$
$$0 * x = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary2(x, y) = binary2(y, x).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, y) = binary1(y, x).
binary1((binary2(x, y)), x) = x.
binary2((binary1(x, y)), x) = x.
binary2(0, x) = x.
binary2(1, x) = 1.
binary1(x, (binary2(y, z))) = binary2((binary1(x, y)), (binary1(x, z))).
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
unary1(0) = 1.
unary1(1) = 0.
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, 0) = 0.
binary1(0, x) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/73`, definition #73 implies definition #94. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.178 (Ortholattices implica Shells)**.** *A classe definida pelos seguintes axiomas (Ortholattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$

$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$
$$x + 0 = x.$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$0 * x = 0.$$
$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/80`, definition #80 implies definition #99. Number of proofs: 6. Max length proof: 10. □

Teorema 4.5.179 (Ortholattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Ortholattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$

$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/80`, definition #80 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Teorema 4.5.180 (Orthomodular lattices implica Shells)**.** *A classe definida pelos seguintes axiomas (Orthomodular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$

$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$
$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

está contida na classe definida pelos seguintes axiomas (Shells)

$$0 + x = x.$$
$$x + 0 = x.$$
$$1 * x = x.$$
$$x * 1 = x.$$
$$0 * x = 0.$$
$$x * 0 = 0.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
binary1(x, (binary2(unary1(x), (binary1(x, y))))) = binary1(x, y).
end_of_list.
formulas(goals).
binary1(0, x) = x.
binary1(x, 0) = x.
binary2(1, x) = x.
binary2(x, 1) = x.
binary2(0, x) = 0.
binary2(x, 0) = 0.
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/81`, definition #81 implies definition #99. Number of proofs: 6. Max length proof: 10. □

Teorema 4.5.181 (Orthomodular lattices implica Normal bands)**.** *A classe definida pelos seguintes axiomas (Orthomodular lattices)*

$$(x \vee y) \vee z = x \vee (y \vee z).$$
$$(x \wedge y) \wedge z = x \wedge (y \wedge z).$$
$$x \vee y = y \vee x.$$
$$x \wedge y = y \wedge x.$$
$$(x \vee y) \wedge x = x.$$
$$(x \wedge y) \vee x = x.$$
$$0 \vee x = x.$$
$$1 \vee x = 1.$$
$$x \vee x' = 1.$$
$$x \wedge x' = 0.$$
$$(x \vee y)' = x' \wedge y'.$$
$$(x \wedge y)' = x' \vee y'.$$
$$x \vee (x' \wedge (x \vee y)) = x \vee y.$$

está contida na classe definida pelos seguintes axiomas (Normal bands)

$$(x * y) * z = x * (y * z).$$
$$x * x = x.$$
$$((x * y) * z) * x = ((x * z) * y) * x.$$

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary2((binary2(x, y)), z) = binary2(x, (binary2(y, z))).
binary1(x, y) = binary1(y, x).
binary2(x, y) = binary2(y, x).
binary2((binary1(x, y)), x) = x.
binary1((binary2(x, y)), x) = x.
binary1(0, x) = x.
binary1(1, x) = 1.
binary1(x, unary1(x)) = 1.
binary2(x, unary1(x)) = 0.
(unary1(binary1(x, y))) = binary2(unary1(x), unary1(y)).
(unary1(binary2(x, y))) = binary1(unary1(x), unary1(y)).
binary1(x, (binary2(unary1(x), (binary1(x, y))))) = binary1(x, y).
end_of_list.
formulas(goals).
binary1((binary1(x, y)), z) = binary1(x, (binary1(y, z))).
binary1(x, x) = x.
binary1((binary1((binary1(x, y)), z)), x) =
binary1((binary1((binary1(x, z)), y)), x).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/definition/81`, definition #81 implies definition #71. Number of proofs: 3. Max length proof: 10. □

Capítulo 5

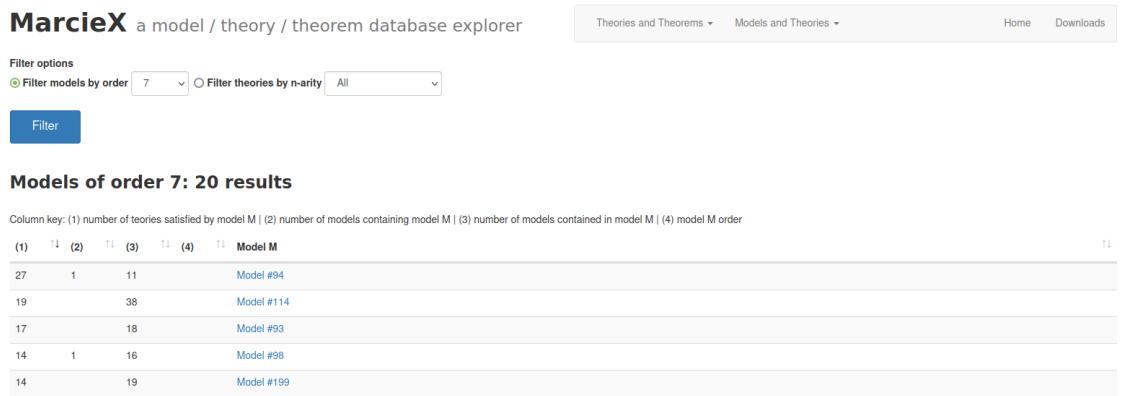
MarcieX

O interface com o utilizador está acessível em `www.marciex.com`. São descritas de seguida as várias opções disponíveis no menu da aplicação.

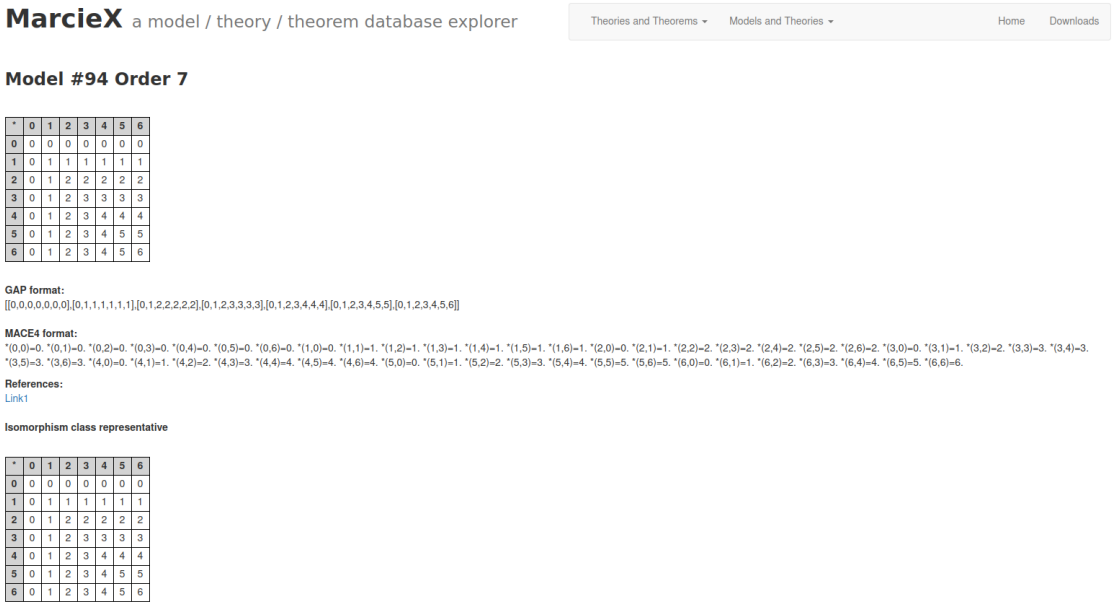
5.1 Models and Theories

Nesta opção o utilizador pode selecionar se pretende visualizar todos os modelos na base de dados ou de uma ordem específica. Também é possível selecionar se pretende visualizar todas as teorias ou apenas de um tipo específico.

Para uma seleção de modelos de ordem 7 vem que



Clicando num link `Model#`, obtém-se detalhes do modelo selecionado, nomeadamente em formato GAP[1], Mace4 e referências na literatura onde foi obtido o modelo. Todos os modelos da base de dados de ordem 7 podem ser consultados no anexo 8.1.



MarcieX a model / theory / theorem database explorer

Theories and Theorems ▾ Models and Theories ▾ Home Downloads

Model #94 Order 7

*	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1
2	0	1	2	2	2	2	2
3	0	1	2	3	3	3	3
4	0	1	2	3	4	4	4
5	0	1	2	3	4	5	5
6	0	1	2	3	4	5	6

GAP format:
[[0,0,0,0,0,0,0],[0,1,1,1,1,1,1],[0,1,2,2,2,2,2],[0,1,2,3,3,3,3],[0,1,2,3,4,4,4],[0,1,2,3,4,5,5],[0,1,2,3,4,5,6]]

MACE4 format:
*(0,0)=0. *(0,1)=0. *(0,2)=0. *(0,3)=0. *(0,4)=0. *(0,5)=0. *(0,6)=0. *(1,0)=0. *(1,1)=1. *(1,2)=1. *(1,3)=1. *(1,4)=1. *(1,5)=1. *(1,6)=1. *(2,0)=0. *(2,1)=1. *(2,2)=2. *(2,3)=2. *(2,4)=2. *(2,5)=2. *(2,6)=2. *(3,0)=0. *(3,1)=1. *(3,2)=2. *(3,3)=3. *(3,4)=3. *(3,5)=3. *(3,6)=3. *(4,0)=0. *(4,1)=1. *(4,2)=2. *(4,3)=3. *(4,4)=4. *(4,5)=4. *(4,6)=4. *(5,0)=0. *(5,1)=1. *(5,2)=2. *(5,3)=3. *(5,4)=4. *(5,5)=5. *(5,6)=5. *(6,0)=0. *(6,1)=1. *(6,2)=2. *(6,3)=3. *(6,4)=4. *(6,5)=5. *(6,6)=6.

References:
Link1

Isomorphism class representative

*	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1
2	0	1	2	2	2	2	2
3	0	1	2	3	3	3	3
4	0	1	2	3	4	4	4
5	0	1	2	3	4	5	5
6	0	1	2	3	4	5	6

Também é possível visualizar modelos que mergulham no modelo ou em que o modelo mergulha, e teorias que o modelo selecionado satisfaz.

1 model containing model #94:
model #43 order 20 File

11 models contained in model #94:
model #8 order 2 File | model #10 order 2 File | model #11 order 3 File | model #173 order 3 File | model #179 order 3 File | model #236 order 3 File | model #237 order 3 File | model #156 order 4 File | model #323 order 4 File | model #617 order 4 File | model #72 order 6 File

Model #94 is model of 27 definitions:
#17 Clifford algebras File | #35 Directoids File | #66 Multiplicative semilattices File | #71 Normal bands File | #74 Order algebras File | #94 Semigroups with zero File | #97 Semirings with zero File | #105 Near-semiring File | #119 Left closure semigroup (LC-semigroup) File | #121 Inverse semigroup File | #122 Domain range semigroup (dr-semigroup) File | #124 Semilattice pseudo-complemented semigroup (SP-semigroup) File | #125 Closable semilattice pseudo-complemented semigroup (Closable SP-semigroup) File | #141 Trisemigroup File | #146 Left disemigroup File | #147 Right disemigroup File | #148 Disemigroup File | #150 Doppelsemigroup File | #151 Strong doppelsemigroup File | #152 Dimonoid File | #153 Commutative dimonoid File | #154 Diassociative semigroup File | #155 Duplicial semigroup File | #157 Associative dialgebra File | #158 Associative trialgebra (Associative trioid) File | #159 Trialgebra (Trioid) File | #160 Associative dimonoid File

É mostrado de seguida um excerto da página 7 do artigo no link da referência[2]

$\times$	L_0	L_1	L_2	L_3	L_4	L_5	L_6
L_0	L_0	L_0	L_0	L_0	L_0	L_0	L_0
L_1	L_0	L_1	L_1	L_1	L_1	L_1	L_1
L_2	L_0	L_1	L_2	L_2	L_2	L_2	L_2
L_3	L_0	L_1	L_2	L_3	L_3	L_3	L_3
L_4	L_0	L_1	L_2	L_3	L_4	L_4	L_4
L_5	L_0	L_1	L_2	L_3	L_4	L_5	L_5
L_6	L_0	L_1	L_2	L_3	L_4	L_5	L_6

Table 2: Multiplication table

[1] GAP – Groups, Algorithms, and Programming. `https://www.gap-system.org`
[2] `https://arxiv.org/pdf/0709.0522.pdf#page=7`

A título de exemplo, apresentada-se ainda um modelo de ordem 6 que aparece mergulhado em dois modelos de ordem 12 e simultaneamente mergulham nele os modelos de ordem 3 e 2 que o precedem. São indicadas as referências do modelo no site, na literatura e classificação usada no *arXiv*. No final, é mostrado um exemplo de ficheiro de entrada em Mace4, que permite demonstrar que o modelo de ordem 6 mergulha no primeiro modelo de ordem 12 apresentado (em notações prefixas).

O modelo de ordem 6

*	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	0	5	4	3	2
2	2	3	4	5	0	1
3	3	2	1	0	5	4
4	4	5	0	1	2	3
5	5	4	3	2	1	0

`https://marciedb.pythonanywhere.com/model/219` ou em:
`https://arxiv.org/pdf/1505.04528.pdf#page=26`
Subjects: Mathematical Physics (math-ph); High Energy Physics - Theory (hep-th); Group Theory (math.GR)

está mergulhado nos modelos de ordem 12:

*	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	2	3	4	5	6	7	8	9	10	11
1	1	0	4	5	2	3	7	6	10	11	8	9
2	2	5	0	4	3	1	8	11	6	10	9	7
3	3	4	5	0	1	2	9	10	11	6	7	8
4	4	3	1	2	5	0	10	9	7	8	11	6
5	5	2	3	1	0	4	11	8	9	7	6	10
6	6	7	8	9	11	10	0	1	2	3	5	4
7	7	6	10	11	9	8	1	0	5	4	2	3
8	8	11	6	10	7	9	2	4	0	5	3	1
9	9	10	11	6	8	7	3	5	4	0	1	2
10	10	9	7	8	6	11	4	2	3	1	0	5
11	11	8	9	7	10	6	5	3	1	2	4	0

`https://marciedb.pythonanywhere.com/model/195` ou em:
`https://arxiv.org/pdf/1507.08455.pdf#page=17`
Subjects: Chaotic Dynamics (nlin.CD); Mathematical Physics (math-ph); Cellular Automata and Lattice Gases (nlin.CG)

*	0	1	2	3	4	5	6	7	8	9	10	11
0	2	3	0	1	6	7	4	5	10	11	8	9
1	4	5	8	9	0	1	10	11	2	3	6	7
2	0	1	2	3	4	5	6	7	8	9	10	11
3	6	7	10	11	2	3	8	9	0	1	4	5
4	8	9	4	5	10	11	0	1	6	7	2	3
5	0	1	2	3	4	5	6	7	8	9	10	11
6	10	11	6	7	8	9	2	3	4	5	0	1
7	2	3	0	1	6	7	4	5	10	11	8	9
8	4	5	8	9	0	1	10	11	2	3	6	7
9	10	11	6	7	8	9	2	3	4	5	0	1
10	6	7	10	11	2	3	8	9	0	1	4	5
11	8	9	4	5	10	11	0	1	6	7	2	3

`https://marciedb.pythonanywhere.com/model/463` ou em:
`https://arxiv.org/pdf/1104.3314.pdf#page=8`
`https://arxiv.org/pdf/1104.3314.pdf#page=10`
Subjects: Formal Languages and Automata Theory (cs.FL); Group Theory (math.GR)

e mergulham no modelo de ordem 6, os modelos de ordem 3:

*	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

`https://marciedb.pythonanywhere.com/model/104` ou em:
`https://arxiv.org/pdf/math-ph/9807012.pdf#page=35`
Subjects: Mathematical Physics (math-ph); High Energy Physics - Theory (hep-th); Quantum Algebra (math.QA)
`https://arxiv.org/pdf/1403.0507.pdf#page=4`
Subjects: Group Theory (math.GR)
`https://arxiv.org/pdf/0909.1065.pdf#page=17`
Subjects: Group Theory (math.GR)

*	0	1	2
0	2	0	1
1	0	1	2
2	1	2	0

`https://marciedb.pythonanywhere.com/model/244` ou em:
`https://arxiv.org/pdf/0909.3747.pdf#page=3`
`https://arxiv.org/pdf/0909.3747.pdf#page=17`
Subjects: General Mathematics (math.GM)

Demonstração. Pode ser obtida a partir da entrada em Mace4, que o modelo de ordem 6 mergulha no modelo #195 de ordem 12:

```
formulas(assumptions).
```

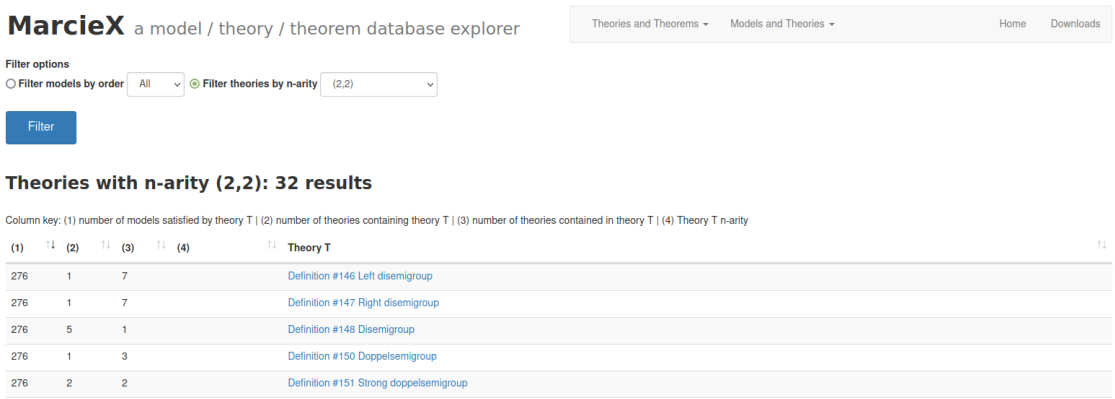
```
% model order 6
% model #219
+(0,0)=0. +(0,1)=1. +(0,2)=2. +(0,3)=3. +(0,4)=4. +(0,5)=5. +(1,0)=1. +(1,1)=0.
+(1,2)=5. +(1,3)=4. +(1,4)=3. +(1,5)=2. +(2,0)=2. +(2,1)=3. +(2,2)=4. +(2,3)=5.
+(2,4)=0. +(2,5)=1. +(3,0)=3. +(3,1)=2. +(3,2)=1. +(3,3)=0. +(3,4)=5. +(3,5)=4.
+(4,0)=4. +(4,1)=5. +(4,2)=0. +(4,3)=1. +(4,4)=2. +(4,5)=3. +(5,0)=5. +(5,1)=4.
+(5,2)=3. +(5,3)=2. +(5,4)=1. +(5,5)=0.

% model order 12
% model #195
*(0,0)=0. *(0,1)=1. *(0,2)=2. *(0,3)=3. *(0,4)=4. *(0,5)=5. *(0,6)=6. *(0,7)=7.
*(0,8)=8. *(0,9)=9. *(0,10)=10. *(0,11)=11. *(1,0)=1. *(1,1)=0. *(1,2)=4. *(1,3)=5.
*(1,4)=2. *(1,5)=3. *(1,6)=7. *(1,7)=6. *(1,8)=10. *(1,9)=11. *(1,10)=8. *(1,11)=9.
*(2,0)=2. *(2,1)=5. *(2,2)=0. *(2,3)=4. *(2,4)=3. *(2,5)=1. *(2,6)=8. *(2,7)=11.
*(2,8)=6. *(2,9)=10. *(2,10)=9. *(2,11)=7. *(3,0)=3. *(3,1)=4. *(3,2)=5. *(3,3)=0.
*(3,4)=1. *(3,5)=2. *(3,6)=9. *(3,7)=10. *(3,8)=11. *(3,9)=6. *(3,10)=7. *(3,11)=8.
*(4,0)=4. *(4,1)=3. *(4,2)=1. *(4,3)=2. *(4,4)=5. *(4,5)=0. *(4,6)=10. *(4,7)=9.
*(4,8)=7. *(4,9)=8. *(4,10)=11. *(4,11)=6. *(5,0)=5. *(5,1)=2. *(5,2)=3. *(5,3)=1.
*(5,4)=0. *(5,5)=4. *(5,6)=11. *(5,7)=8. *(5,8)=9. *(5,9)=7. *(5,10)=6. *(5,11)=10.
*(6,0)=6. *(6,1)=7. *(6,2)=8. *(6,3)=9. *(6,4)=11. *(6,5)=10. *(6,6)=0. *(6,7)=1.
*(6,8)=2. *(6,9)=3. *(6,10)=5. *(6,11)=4. *(7,0)=7. *(7,1)=6. *(7,2)=10. *(7,3)=11.
*(7,4)=9. *(7,5)=8. *(7,6)=1. *(7,7)=0. *(7,8)=5. *(7,9)=4. *(7,10)=2. *(7,11)=3.
*(8,0)=8. *(8,1)=11. *(8,2)=6. *(8,3)=10. *(8,4)=7. *(8,5)=9. *(8,6)=2. *(8,7)=4.
*(8,8)=0. *(8,9)=5. *(8,10)=3. *(8,11)=1. *(9,0)=9. *(9,1)=10. *(9,2)=11. *(9,3)=6.
*(9,4)=8. *(9,5)=7. *(9,6)=3. *(9,7)=5. *(9,8)=4. *(9,9)=0. *(9,10)=1. *(9,11)=2.
*(10,0)=10. *(10,1)=9. *(10,2)=7. *(10,3)=8. *(10,4)=6. *(10,5)=11. *(10,6)=4.
*(10,7)=2. *(10,8)=3. *(10,9)=1. *(10,10)=0. *(10,11)=5. *(11,0)=11. *(11,1)=8.
*(11,2)=9. *(11,3)=7. *(11,4)=10. *(11,5)=6. *(11,6)=5. *(11,7)=3. *(11,8)=1.
*(11,9)=2. *(11,10)=4. *(11,11)=0.
f(x) = f(y) -> x = y.
f(x + y) = f(x) * f(y).
end_of_list.
formulas(goals).
end_of_list.
```

Ou em `https://marciedb.pythonanywhere.com/model/219`, 2 models containing model #219. □

Observe-se como estes modelos aparecem em áreas muito diferentes da matemática e como todas estas considerações poderão ajudar a identificar ligações esquivas até agora.

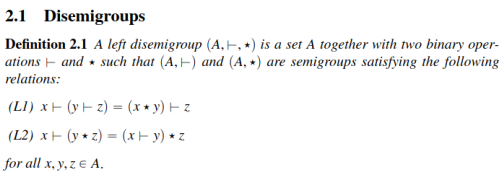
Por outro lado, para uma seleção de teorias do tipo (2,2) vem que



Clicando num link `Definition#`, obtém-se detalhes da teoria selecionada, nomeadamente em formato prefixo e referências na literatura onde foi obtida a teoria.

Além das referências de modelos que satisfazem a teoria, também é possível consultar teorias contidas ou que contêm a teoria selecionada.

É mostrado de seguida um excerto da página 2 do artigo no link da referência[3]



2.1 Disemigroups

Definition 2.1 *A left disemigroup $(A, \vdash, \star)$ is a set A together with two binary operations $\vdash$ and $\star$ such that $(A, \vdash)$ and $(A, \star)$ are semigroups satisfying the following relations:*

(L1) $x \vdash (y \vdash z) = (x \star y) \vdash z$

(L2) $x \vdash (y \star z) = (x \vdash y) \star z$

for all $x, y, z \in A$.

Os modelos apresentados de seguida de ordem 8 e de ordem 4 satisfazem o conjunto de axiomas que os precedem[4]. São indicadas as referências do site, da literatura e classificação usada no *arXiv*. No final, é mostrado um exemplo de ficheiro de entrada em Mace4, que permite demonstrar que o modelo de ordem 8 (em notação prefixa) satisfaz o conjunto de axiomas apresentado.

[3] `https://arxiv.org/pdf/1904.12030.pdf#page=2`

[4] O modelo de ordem 8 contém por sua vez o modelo de ordem 4.

Os seguintes modelos de ordem 8 e 4

*	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	2	0	0	2	0	2
3	0	3	0	3	4	6	6	4
4	0	4	0	4	4	0	0	4
5	0	5	2	6	0	5	6	2
6	0	6	0	6	0	6	6	0
7	0	7	2	4	4	2	0	7

`https://marciedb.pythonanywhere.com/model/70` ou em:
`https://arxiv.org/pdf/quant-ph/0605239.pdf#page=10`
Subjects: Quantum Physics (quant-ph); Mathematical Physics (math-ph)

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	2	0
3	0	3	0	3

`https://marciedb.pythonanywhere.com/model/65` ou em:
`https://arxiv.org/pdf/quant-ph/0603051.pdf#page=3`
Subjects: Quantum Physics (quant-ph); Mathematical Physics (math-ph)
satisfazem o seguinte conjunto de axiomas *(Antidomain monoid)*

$$(x * y) * z = x * (y * z).$$

$$1 * x = x.$$

$$x * 1 = x.$$

$$x' * x = 0.$$

$$x * 0 = 0.$$

$$x' * y' = y' * x'.$$

$$x'' * x = x.$$

$$x' = (x * y)' * (x * y')'.$$

$$(x * y)' * x = ((x * y)' * x) * y'.$$

Demonstração. Pode ser obtida a partir da entrada em Mace4, para o modelo de ordem 8:

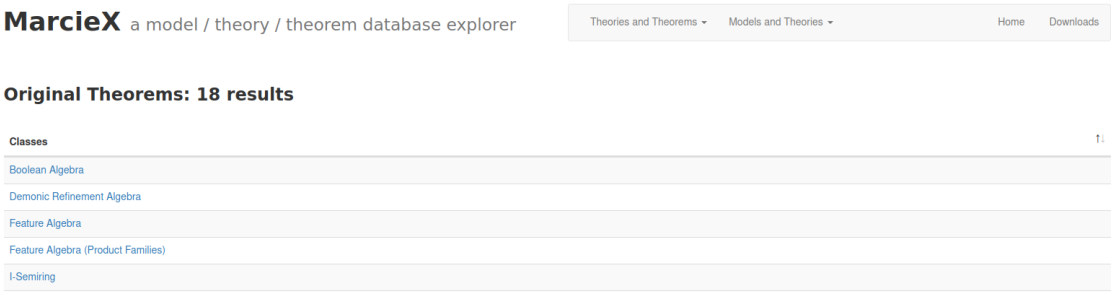
```
formulas(assumptions).
% definition #123 Antidomain monoid(a-monoid)
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
```

```
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% model order 8
% model #70
*(0,0)=0. *(0,1)=0. *(0,2)=0. *(0,3)=0. *(0,4)=0. *(0,5)=0. *(0,6)=0. *(0,7)=0.
*(1,0)=0. *(1,1)=1. *(1,2)=2. *(1,3)=3. *(1,4)=4. *(1,5)=5. *(1,6)=6. *(1,7)=7.
*(2,0)=0. *(2,1)=2. *(2,2)=2. *(2,3)=0. *(2,4)=0. *(2,5)=2. *(2,6)=0. *(2,7)=2.
*(3,0)=0. *(3,1)=3. *(3,2)=0. *(3,3)=3. *(3,4)=4. *(3,5)=6. *(3,6)=6. *(3,7)=4.
*(4,0)=0. *(4,1)=4. *(4,2)=0. *(4,3)=4. *(4,4)=4. *(4,5)=0. *(4,6)=0. *(4,7)=4.
*(5,0)=0. *(5,1)=5. *(5,2)=2. *(5,3)=6. *(5,4)=0. *(5,5)=5. *(5,6)=6. *(5,7)=2.
*(6,0)=0. *(6,1)=6. *(6,2)=0. *(6,3)=6. *(6,4)=0. *(6,5)=6. *(6,6)=6. *(6,7)=0.
*(7,0)=0. *(7,1)=7. *(7,2)=2. *(7,3)=4. *(7,4)=4. *(7,5)=2. *(7,6)=0. *(7,7)=7.
end_of_list.
formulas(goals).
end_of_list.
```

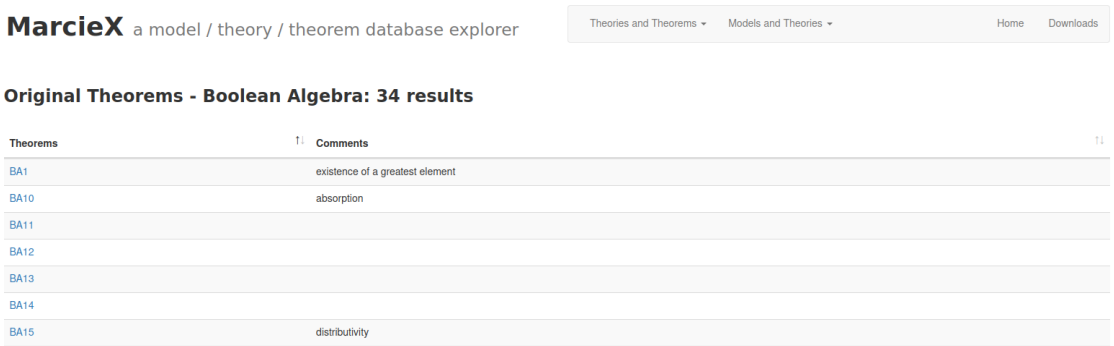
Ou em `https://marciedb.pythonanywhere.com/definition/123`, definition #123 Antidomain monoid (a-monoid) have 12 models. □

5.2 Original theorems

Nesta opção o utilizador pode visualizar todas as classes de teoremas originais da base de dados. O conjunto de todos os teoremas constitui o eixo dos XX do plano, conforme descrito na introdução desta tese. Este conjunto de teoremas é denominado por teoremas geradores (secção 6.1).



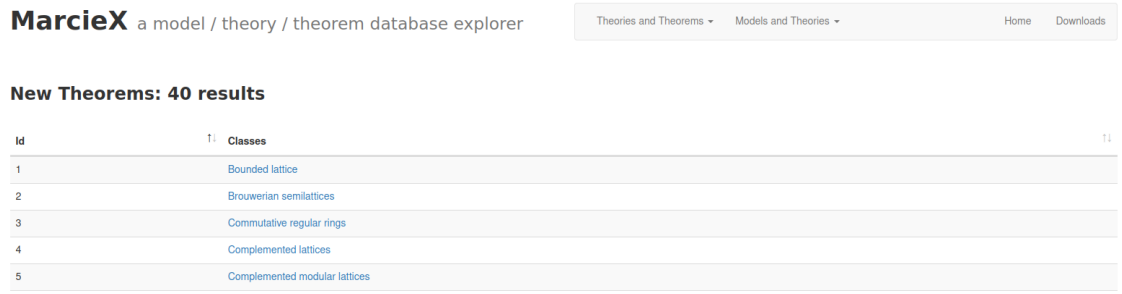
Clicando por exemplo no link `Boolean Algebra`, obtém-se todos os teoremas da classe selecionada.



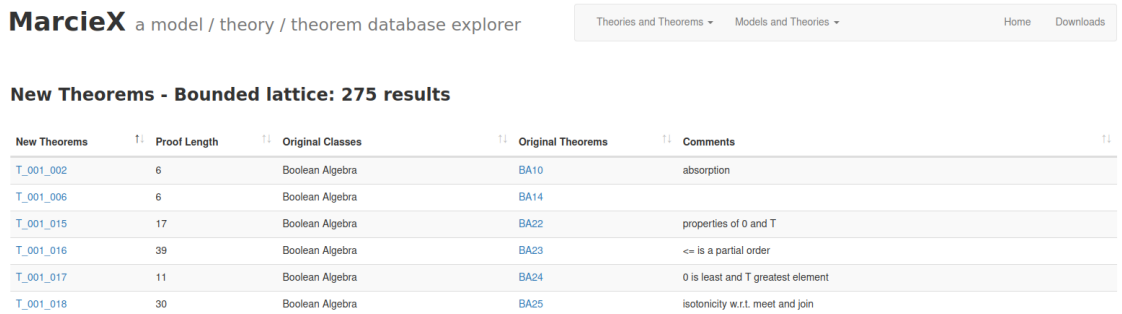
Clicando por exemplo no link `BA1`, obtém-se detalhes do teorema selecionado, como referências na literatura, ficheiro do teorema em formato Prover9 e respetivo ficheiro da prova.

5.3 Generated theorems

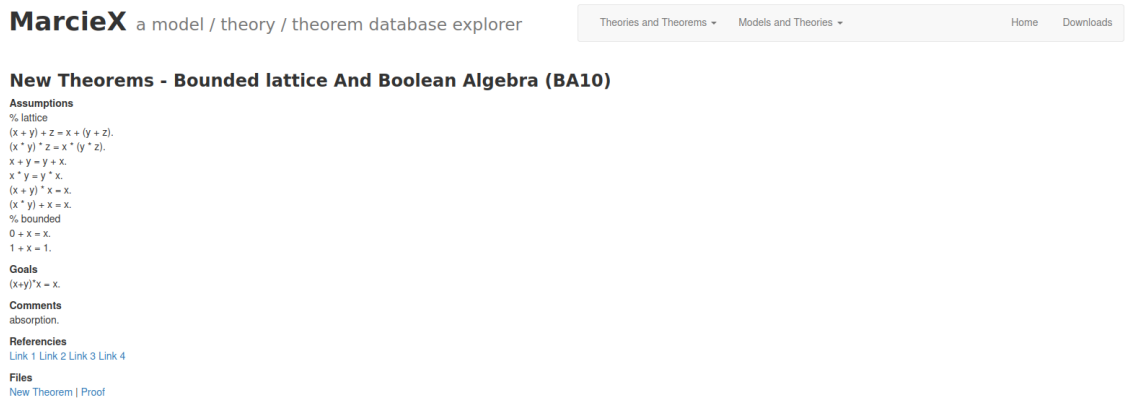
Nesta opção o utilizador pode visualizar todas as classes de teorias originais da base de dados. O conjunto de todas as teorias constitui o eixo dos YY do plano, conforme descrito na introdução desta tese.



Clicando por exemplo no link `Bounded lattice`, obtém-se todos os teoremas gerados para a classe selecionada. Estes teoremas gerados constituem os pontos do plano (T_j, V_i), em que o teorema T_j é verdadeiro na teoria V_i. Este conjunto de teoremas é denominado por teoremas gerados (secção 6.2).

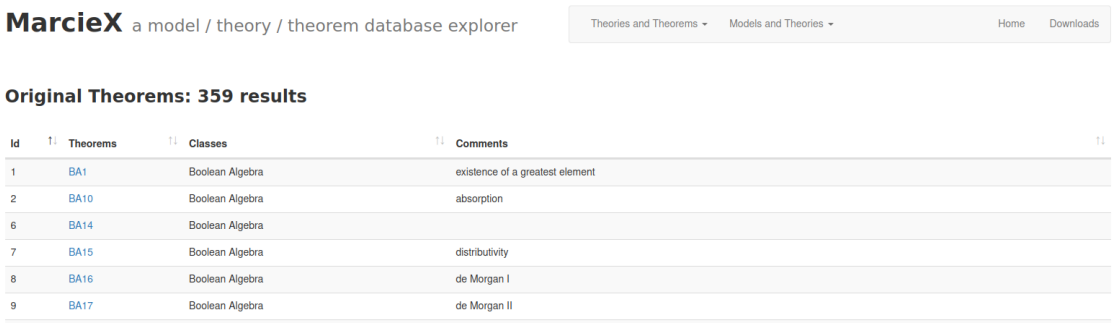


Clicando por exemplo no link `T_001_002`, obtém-se detalhes do novo teorema selecionado, como referências na literatura, ficheiro do novo teorema em formato Prover9 e respetivo ficheiro da prova.

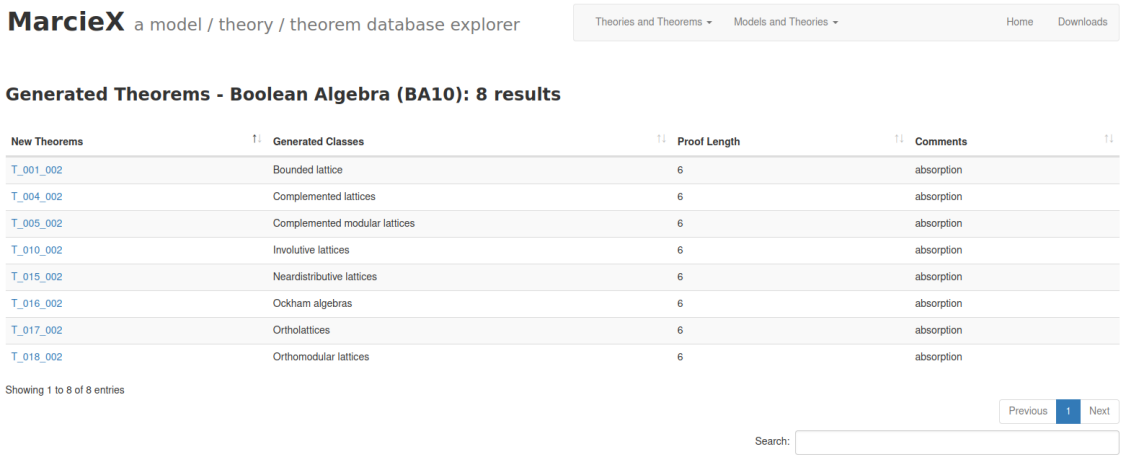


5.4 Original / Generated theorems

Nesta opção o utilizador pode visualizar todos os teoremas novos gerados a partir de um dado teorema gerador.

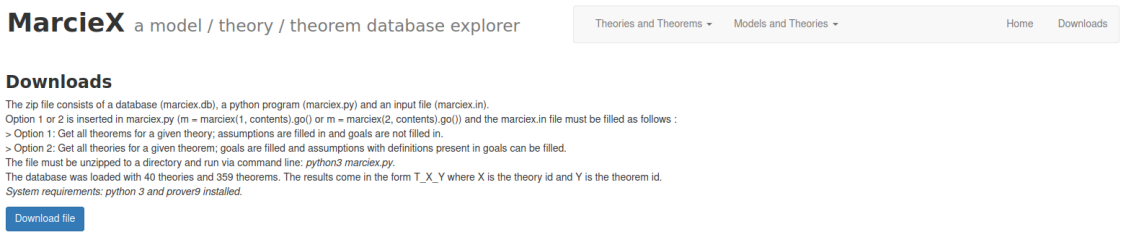


Clicando por exemplo no link `B10`, obtém-se todos os teoremas gerados para um dado teorema gerador.



5.5 Download de aplicação

Um utilizador que descobre um teorema T numa axiomática A, pode colocar T no nosso sistema que irá procurar pares (X, T), quando X percorre a base de dados de axiomáticas do nosso repositório, tendo como resposta (A_1, T), (A_2, T),..., (A_n, T). De forma análoga, um matemático pode ser levado à introdução de uma nova classe A. O nosso sistema vai procurar pares (A, X), onde X são teoremas na base de dados, pelo que poucos minutos depois de ter identificado a nova classe, o utilizador poderá já ter vários teoremas familiares, possivelmente não triviais e uma base muito mais sólida para prosseguir a sua investigação. Dadas as limitações descritas na secção 3.4, é disponibilizado um programa que pode ser executado no computador do utilizador.



O ficheiro zip é constituído por uma base de dados SQLite `marciex.db`, um programa escrito em Python 3 `marciex.py` e um ficheiro de entrada `marciex.in`. No programa `marciex.py` é possível selecionar a opção desejada:

- Opção 1: Obtém todos os teoremas de uma dada teoria (As `assumptions` são preenchidas e os `goals` não são preenchidos no ficheiro `marciex.in`);
- Opção 2: Obtém todas as teorias de um dado teorema (Os `goals` são preenchidos e as `assumptions` podem ser preenchidas se contiverem definições necessárias para os `goals` no ficheiro `marciex.in`).

As opções são inseridas em `marciex.py` respetivamente com `m = marciex(1, contents).go()` ou `m = marciex(2, contents).go()`. O ficheiro zip deve ser descompactado para uma diretoria e executado via linha de comando `python3 marciex.py`. Os resultados vem na forma `T_X_Y`

onde X é a identificação da teoria e Y é a identificação do teorema. Para o programa ser executado é necessário que esteja instalado no computador do utilizador o Python3 e o Prover9.

5.6 Model Explorer

A opção Model Explorer permite o acesso a um motor de busca de magmas M (conjunto com uma operação binária) na literatura. O MarcieX permite identificar não só modelos que estão contidos, que contém ou que são isomorfos a M, mas também modelos que satisfazem um dado conjunto de axiomas. Nas páginas seguintes é efetuada uma descrição completa do motor de busca.

MARCIEX - A MODEL SEARCH ENGINE

JOÃO ARAÚJO AND JOÃO RAMIRES

ABSTRACT. MarcieX is a search engine for magmas M (a set with a binary operation) in the literature. MarcieX identifies papers in which appear models containing, contained or isomorphic to M. It also accepts a set of first order sentences and returns papers containing models that satisfy all the sentences.

1. INTRODUCTION

For a small natural n, there are relatively few groups of order n. For example, there are 19 groups of order at most 10 (up to isomorphism) and hence each one of them (and of much larger orders) is well known, has a name, etc. If a group of order 8 is a counter-example for some claim, then it is very likely that everything needed to understand its relation with the conjecture is obvious from our knowledge of that group.

The situation changes dramatically for semigroups, quasigroups, magmas in general, or many other classes of algebras. There are 12,418,001,077,381,302,684 order 10 semigroups up to isomorphism or anti-isomorphism [5]; the number of order 10 quasigroups takes more digits:

$$2,750,892,211,809,150,446,995,735,533,513.$$

A formula for the non-isomorphic order n magmas has been found long ago [7] and yields numbers much larger.

True that these numbers can be very large, but the interest of the models is not equally spread. For example, it has been conjectured (the proof in [8] has an unsolved gap) that the overwhelming majority of finite semigroups have one and only one idempotent which is a zero, denoted 0, and satisfy the identity $xyz = 0$, at first glance a not very exciting variety. Regarding the remaining, when one stumbles upon one small semigroup, important for some reason, (unlike what happens with groups) its properties might be elusive and/or its importance to other parts of mathematics not readily obvious.

Therefore it seems useful to have a search engine to find magmas in the literature as hopefully an important model for someone, might already have been important to someone else and hence there might be some literature about it. Of course, it might happen that the model we are interested on never appeared before; but it might happen that it already appeared and hence the user can be put on the right track to bibliography pointers, possibly unearth interconnections to other parts of mathematics, find properties of the model that others identified, etc. MarcieX [9] does precisely that.

In addition, MarcieX accepts a set of first order sentences and returns papers containing models that satisfy all the sentences. Therefore, if someone finds an interesting class of mathematical objects and wants to find in the literature potentially relevant examples of that theory, then MarcieX is there to help.

2. THE INTERFACE FOR MODELS

The interface to give MarcieX a magma is very flexible. It is possible to feed MarcieX with the Cayley Table of a magma in the universe $\{0, \ldots, n-1\}$ or in the universe $\{1, \ldots, n\}$. For example, the order 4 model M in Figure 1 (which will be our foregoing toy model) can also be given as

$$0, 3, 1, 2, 2, 1, 3, 0, 3, 0, 2, 1, 1, 2, 0, 3$$

or in GAP [6] format as $[[1, 4, 2, 3], [3, 2, 4, 1], [4, 1, 3, 2], [2, 3, 1, 4]]$. MarcieX identifies papers containing models that are isomorphic to the given model M; it also searches for models contained or containing it as shown on Figure 1. MarcieX also provides other information such as isomorphism class representative that we now explain. The vector of an order n magma A, denoted $\vec{v}(A)$,

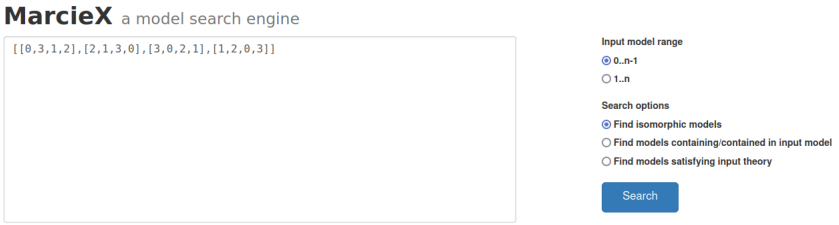


FIGURE 1. Input model M and the *Find isomorphic models* option

is the $n \times n$ vector resulting from the concatenation of the n rows of the multiplication table of A. (The vector $\vec{v}(M)$ of the input model in Figure 1 is $[0, 3, 1, 2, 2, 1, 3, 0, 3, 0, 2, 1, 1, 2, 0, 3]$.) Let $I := \{A_j \mid j \in J\}$ be the set of models isomorphic to the given magma A (in the same universe as A), with $\vec{v}(A_j)$ denoting the vector corresponding to the magma $A_j \in I$. Clearly the vectors in I can be lexicographically ordered. The minimum of I (under the lexicographic order) is considered the canonical representative magma of the isomorphism class I. For example, Table 1 shows the canonical representative of the isomorphism class of M. MarcieX's output claims that the exact

*	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

*	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

TABLE 1. Model M and the representative of its isomorphism class.

model M appears in five articles (Figure 2), accessible through the links provided. The models found are reproduced in Table 2; it also gives the isomorphism class representative of M and provides pointers to two other isomorphic magmas. It is worth observing that these magmas appear in papers of very different areas of mathematics such as *geometry of quantum groups*, *knot theory*, *Hopf algebras*, *AG-groupoids*, etc. To ease further research, all these magmas are returned in GAP format and *cooked prefix format* to be used in Mace4 [10]. The search for magmas contained or containing the given magma is similar.

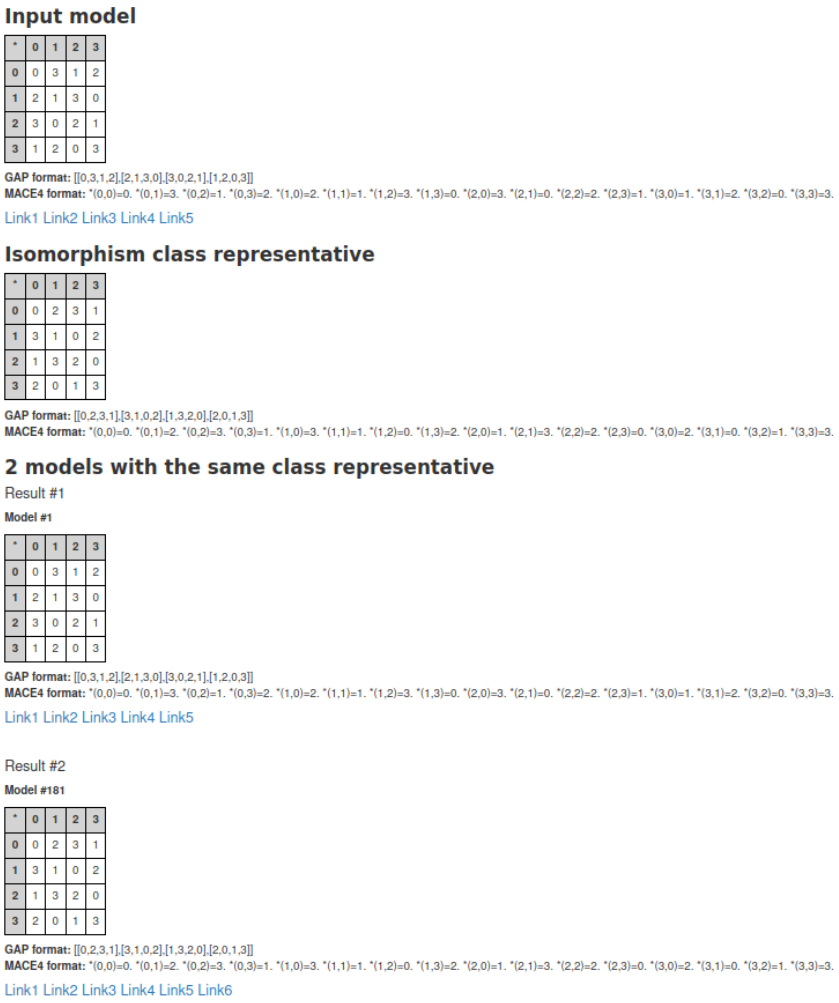


FIGURE 2. MarcieX search model result

▷	1	2	3	4
1	1	4	2	3
2	3	2	4	1
3	4	1	3	2
4	2	3	1	4

row ◁ col	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

R*C	[0]	[1]	[2]	[3]
[0]	[0]	[3]	[1]	[2]
[1]	[2]	[1]	[3]	[0]
[2]	[3]	[0]	[2]	[1]
[3]	[1]	[2]	[0]	[3]

x*y	a	b	c	d
a	a	d	b	c
b	c	b	d	a
c	d	a	c	b
d	b	c	a	d

Ad	t	x	y	z
t	t	z	x	y
x	y	x	z	t
y	z	t	y	x
z	x	y	t	z

TABLE 2. Five representations of model M found in literature.

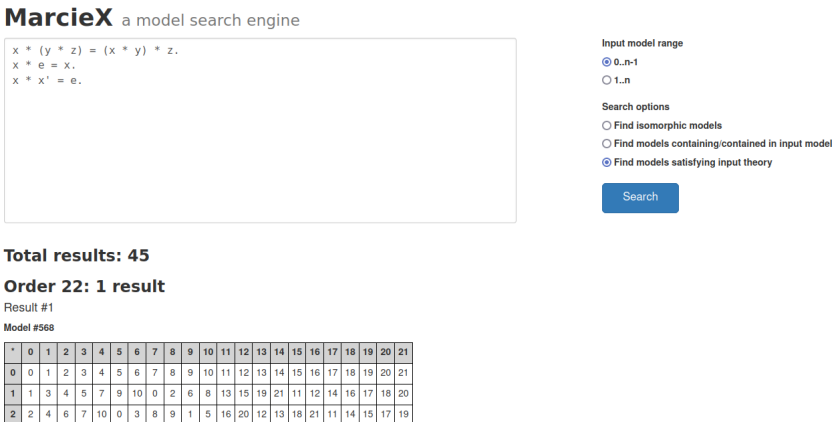
3. THE INTERFACE FOR FOL SENTENCES

To feed MarcieX with a set of sentences in first order logic (FOL) we need the syntax defined in Table 3. Some examples are described in Table 4.

MarcieX is a web application developed in the Python programming language. Uses a model database extracted from thousands of articles (including all Cayley tables from all papers from the arXiv [4]) coupled with some automated reasoning tools. This database will be permanently updated and with the increasing quality of tools to handle images, many tables from old papers will

Symbol	in MarcieX	Observations
$\vee$	`|`	
$\wedge$	`&`	
$\Rightarrow$	`->`	no spaces between $-$ and $>$
$\Leftrightarrow$	`<->`	no spaces between $<$, $-$ and $>$
$\sim$	`~`	negation
$\exists$	`exists`	delimit with brackets the expression affected by the quantifier
$\forall$	`all`	delimit with brackets the expression affected by the quantifier
$\neq$	`!=`	no spaces between ! and $=$
	`.`	every sentence ends with a final mark
	`x,y,...`	are variables (assumed to be universally quantified)
	`a,b,...`	are constants

TABLE 3. MarcieX syntax



*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	3	4	5	7	9	10	0	2	6	8	13	15	19	21	11	12	14	16	17	18	20
2	2	4	6	7	10	0	3	8	9	1	5	16	20	12	13	18	21	11	14	15	17	19

FIGURE 3. Input theory G and the corresponding output

Property	in MarcieX

Associativity of a binary operation
$(\forall x, y, z \in A)\ x(yz) = (xy)z$
	`all x all y all z (x * (y * z) = (x * y) * z).`
	Assumed to be universally quantified, then:
	`x * (y * z) = (x * y) * z.`

$(\exists a, b, c \in A)\ a(bc) = (ab)c$	`a * (b * c) = (a * b) * c.`
Injectivity of $f : A \to A$ $(\forall x, y \in A)\ (f(x) = f(y) \to x = y)$	`f(x) = f(y) -> x = y.`

Let G be a set of defining identities for groups	`x * (y * z) = (x * y) * z.`
	`x * e = x.`
	`x * x' = e.`

TABLE 4. MarcieX syntax examples

be added. To find the representative of the isomorphism class we use the algorithm implemented in [2]. More details can be found in [3].

In addition, the user can insert a set of axioms (see Table 3) and MarcieX will output models in the literature that satisfy the formulas. MarcieX only deals with magmas (therefore, leaving out algebras with n-ary operations for $n \geq 3$) as that covers the most popular algebras. Figure 3 has the example of one theory G (which is the variety of groups defined by three identities) and models satisfying G.

Input theories that handle multiple binary operations will return models that satisfy all binary operations simultaneously. For example, with an input theory with three binary operations, the

same model is checked for all three operations. (If all model combinations were checked for all three operations, the runtime would be extremely time consuming –but we will try to add that feature in the future–.)

We wrote a version of MarcieX to run over ProverX [9, 10], a scriptable automated theorem prover with an integrated development environment that runs in the cloud. The advanced user can take advantage of that interactive format that is not possible on the website version of MarcieX.

ACKNOWLEDGMENT

MarcieX needs to be permanently updated. You can help us in two ways:

(1) whenever you have a paper containing an *interesting* model, please send it to us in order to be added to the database (if your paper is on the arXiv, there is no need to send as it will be automatically added);

(2) it is possible that you use this site to get insight, find connections to other parts of mathematics, etc., but in the end all results stand by themselves so you see no need to cite MarcieX. Nevertheless, please cite it since without citations funding bodies will not find it useful and without fundings it will not be possible to keep updating.

REFERENCES

[1] João Araújo, Michael Kinyon and Yves Robert, Varieties of regular semigroups with uniquely defined inversion. Port. Math. 76 (2019), 205-228.

[2] João Araújo, João Pedro Araújo, Peter J. Cameron, Edmond W. H. Lee and Jorge Raminhos, Varieties of Semigroups, 2019, `https://sgv.pythonanywhere.com`

[3] João Araújo, João Pedro Araújo, Peter J. Cameron, Edmond W. H. Lee and Jorge Raminhos, A Survey on Varieties Generated by Small Semigroups and a Companion Website, 2019, Preprint `https://arxiv.org/abs/1911.05817`

[4] arXiv, an open-access archive for scholarly articles `https://arxiv.org`

[5] Andreas Distler, Chris Jefferson, Tom Kelsey and Lars Kotthoff, The Semigroups of Order 10, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science book series (LNCS) **7514**, 883–899.

[6] The GAP Group, *GAP – Groups, Algorithms, and Programming*, 2020, `https://www.gap-system.org`

[7] Michael A. Harrison, The number of isomorphism types of finite algebras, *Proc. Amer. Math. Soc.* **17** (1966), 731–737.

[8] D. J. Kleitman, B. R. Rothschild and J. H. Spencer, The number of semigroups of order n, *Proceedings of the American Mathematical Society*, **55**, 1, (1976), 227–232.

[9] MarcieX, a model search engine, 2021, `https://marciex.pythonanywhere.com`

[10] William McCune, Prover9/Mace4, version 2009-11A, `http://www.cs.unm.edu/~mccune/prover9`

[11] ProverX, an automated theorem prover tool, 2020, `www.proverx.com`

MATHEMATICS DEPARTMENT, FACULTY OF SCIENCE AND TECHNOLOGY, UNIVERSIDADE NOVA DE LISBOA, PORTUGAL

Email address: `jj.araujo@fct.unl.pt`

Email address: `joao.j.ramires@gmail.com`

MATHEMATICS DEPARTMENT, UNIVERSIDADE ABERTA, LISBON, PORTUGAL

Capítulo 6

Teoremas

Neste capítulo são apresentados teoremas geradores e teoremas gerados, obtidos da forma descrita na secção 4.1. Uma vez que usamos muitas classes de objetos matemáticos, a maioria sem tradução oficial para português, optámos por usar o nome técnico inglês de forma a ser claro qual a estrutura em causa. De igual modo, quando nos referimos a um resultado na literatura (lema, proposição, teorema) usamos a classificação que consta no original (Lemma, Proposition, etc) de forma a garantir uma identificação clara do resultado.

6.1 Teoremas geradores

6.1.1 Semigroups

As seguintes proposições, lemas e teoremas (referências AS2022_nn) foram obtidos a partir do livro *Fundamentals of Semigroup Theory* da autoria de John Howie [6]. A tradução para sintaxe do Prover9 foi realizado por Afonso Simplício (Departamento de Matemática da Universidade Nova de Lisboa).

Proposição 6.1.1 (AS2022_0)**.** *[6] Proposition 2.1.3.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.1.1 (AS2022_1). *[6] Lemma 2.1.1*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.1.2 (AS2022_2). *[6] Lemma 2.1.1*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.1.3 (AS2022_3). *[6] Lemma 2.1.2*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.1.4 (AS2022_4). *[6] Lemma 2.1.2*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.1.2 (AS2022_5). *[6] Proposition 2.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.1.1 (AS2022_6). *[6] Theorem 2.3.5.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.1.2 (AS2022_8). *[6] Theorem 2.3.5.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.1.3 (AS2022_9). *[6] Theorem 2.3.5.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.1.4 (AS2022_10). *[6] Theorem 2.3.5.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
```

```
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.1.5 (AS2022_11). *[6] Theorem 2.3.5.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.1.6 (AS2022_12). *[6] Theorem 2.3.5.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
```

```
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

6.1.2 Inverse semigroup

Os seguintes teoremas, proposições e lemas (referências IJ2022_nn) foram obtidos a partir do livro *Fundamentals of Semigroup Theory* da autoria de John Howie [6]. A tradução para sintaxe do Prover9 foi realizada por Inês Jesus (Departamento de Matemática da Universidade Nova de Lisboa).

Teorema 6.1.7 (IJ2022_1). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.1.8 (IJ2022_2). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.1.9 (IJ2022_2). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.1.10 (IJ2022_2)**.** *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.1.11 (IJ2022_2)**.** *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.1.12 (IJ2022_3). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.1.13 (IJ2022_4). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.1.14 (IJ2022_5). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.1.15 (IJ2022_6). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.1.16 (IJ2022_6)**.** *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.1.17 (IJ2022_7)**.** *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
```

```
end_of_list.
```

□

Teorema 6.1.18 (IJ2022_8). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.1.19 (IJ2022_10). *[6] Theorem 5.1.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.1.3 (IJ2022_13). *[6] Proposition 5.1.4.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
```

```
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.1.4 (IJ2022_14)**.** *[6] Proposition 5.1.4.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.1.5 (IJ2022_15)**.** *[6] Proposition 5.1.4.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.1.6 (IJ2022_16)**.** *[6] Proposition 5.1.4.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.1.7 (IJ2022_18). *[6] Proposition 5.1.4.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%semigroup
(x * y) * z = x * (y * z).
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.1.8 (IJ2022_19). *[6] Proposition 5.1.6.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
%(T,+) is a semigroup
(x + y) + z = x + (y + z).
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Proposição 6.1.9 (IJ2022_19). *[6] Proposition 5.1.6.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
%(T,+) is a semigroup
(x + y) + z = x + (y + z).
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Proposição 6.1.10 (IJ2022_20). *[6] Proposition 5.1.6.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
%(T,+) is a semigroup
(x + y) + z = x + (y + z).
% IJ2022_20
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
% inverse are unique
H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y)) -> H(y) = H(x').
end_of_list.
```

□

Lema 6.1.5 (IJ2022_21). *[6] Lemma 5.1.11*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.1.6 (IJ2022_22). *[6] Lemma 5.1.11*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
```

```
exists b (c = b * f).
end_of_list.
```

□

Lema 6.1.7 (IJ2022_23). *[6] Lemma 5.2.1*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.1.8 (IJ2022_24). *[6] Lemma 5.2.1*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.1.9 (IJ2022_25). *[6] Lemma 5.2.1*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.1.11 (IJ2022_26). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_26
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
R(a,b).
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a * a' = a * b'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.1.12 (IJ2022_27). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_27
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
a * a' = b * a'.
a * a' = a * b'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.1.13 (IJ2022_28). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_28
```

```
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = b * a'.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = a * b'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.1.14 (IJ2022_29). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_29
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = a * b'.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.1.15 (IJ2022_30). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_30
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = b' * a.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a * a' = a * b'.
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.1.16 (IJ2022_31). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_31
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = a' * b.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a * a' = a * b'.
a' * a = b' * a.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.1.17 (IJ2022_32). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_32
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * b') * a.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a * a' = a * b'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.1.18 (IJ2022_33). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_33
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
```

```
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a * a' = a * b'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * b') * a.
end_of_list.
```

□

Proposição 6.1.19 (IJ2022_34). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_34
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.1.20 (IJ2022_35). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_35
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = b * a'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.1.21 (IJ2022_36). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_36
```

```
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = a * b'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.1.22 (IJ2022_37). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_37
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = b' * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.1.23 (IJ2022_38). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_38
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = a' * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.1.24 (IJ2022_39). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_39
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
```

```
a = (a * b') * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.1.25 (IJ2022_40). *[6] Proposition 5.2.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_40
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.1.26 (IJ2022_41). *[6] Proposition 5.2.4.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.1.27 (IJ2022_41A). *[6] Proposition 5.2.8.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% inverse semigroup S axioms
(x * y) * z = x * (y * z).   % associativity
x = x * (x' * x) . %regularity
x' = x' * (x * x').
x = x * (y * x ) & y = y *(x * y) -> x' = y. %unique inverse
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
```

```
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.1.20 (IJ2022_42). *[6] Theorem 5.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.1.21 (IJ2022_43). *[6] Theorem 5.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.1.22 (IJ2022_44). *[6] Theorem 5.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.1.23 (IJ2022_45)**.** *[6] Theorem 5.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.1.24 (IJ2022_46)**.** *[6] Theorem 5.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.1.25 (IJ2022_47)**.** *[6] Theorem 5.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.1.26 (IJ2022_48)**.** *[6] Theorem 5.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.1.27 (IJ2022_49)**.** *[6] Theorem 5.3.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
```

```
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.1.28 (IJ2022_50). *[6] Theorem 5.3.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.1.29 (IJ2022_51). *[6] Theorem 5.3.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.1.30 (IJ2022_52). *[6] Theorem 5.3.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.1.31 (IJ2022_53). *[6] Theorem 5.3.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.1.32 (IJ2022_54). *[6] Theorem 5.3.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.1.33 (IJ2022_55). *[6] Theorem 5.3.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.1.34 (IJ2022_56). *[6] Theorem 5.3.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.1.35 (IJ2022_57). *[6] Theorem 5.3.2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%S is an inverse semigroup
(x * y) * z = x * (y * z).
x = (x * x') * x.
x'' = x.
(x * x') * (y * y') = (y * y') * (x * x').
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

6.1.3 Conjugacy

Os seguintes lemas (referências PA2022_n) foram obtidos a partir do artigo *Conjugation in a group* da autoria de Keith Conrad [7]. A tradução para sintaxe do Prover9 foi realizado por Pedro Almeida (Departamento de Ciências e Tecnologia da Universidade Aberta).

Lema 6.1.10 (PA2022_1). *[7] Lemma 3.1, n = 2, Group.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x * (y * z) = (x * y) * z.
x * e = x.
x * x' = e.
% PA2022_1
```

```
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.1.11 (PA2022_2). *[7] Lemma 3.1, n = 3, Group.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x * (y * z) = (x * y) * z.
x * e = x.
x * x' = e.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.1.4 Diassociative automorphic loop

O seguinte teorema (referência MKL1) foi fornecido pelo Professor Michael Kinyon (University of Denver). Pode ser consultado no artigo *Every Diassociative A-Loop is Moufang* da autoria de Michael K. Kinyon, Kenneth Kunen and J. D. Phillips [24].

Teorema 6.1.36 (MKL1). *[24] Every diassociative automorphic loop is a Moufang loop.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
assign(eq_defs,fold).

formulas(assumptions).
% IP loop
1 * x = x. x * 1 = x.
x' * (x * y) = y.
(x * y) * y' = x.
% MKL1
% inner mappings
T(x,y) = y' * (x * y).
R(x,y,z) = ((x * y) * z) * (y * z)'.
L(x,y,z) = (z * y)' * (z * (y * x)).

% automorphic
T(x,y) * T(z,y) = T(x * z,y).
R(x,y,z) * R(u,y,z) = R(x * u,y,z).
L(x,y,z) * L(u,y,z) = L(x * u,y,z).
end_of_list.
formulas(goals).
((x * y) * x) * z = x * (y * (x * z)).
end_of_list.
```

□

6.1.5 F-quasigroup

O seguinte teorema (referência MKL3) foi fornecido pelo Professor Michael Kinyon (University of Denver). Pode ser consultado no artigo *The structure of F-quasigroups* da autoria de Tomávs Kepka, Michael K. Kinyon and J. D. Phillips [25].

Teorema 6.1.37 (MKL3)**.** *[25] Every F-quasigroup is isotopic to a Moufang loop.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).

formulas(assumptions).
% quasigroup axioms
x \ (x * y) = y. x * (x \ y) = y.
(x * y) / y = x. (x / y) * y = x.
(x / y) \ x = y. x / (y \ x) = y.

% F-quasigroup
(x * y) * ((x \ x) * z) = x * (y * z).
(z * (x / x)) * (y * x) = (z * y) * x.
% MKL3

end_of_list.
formulas(goals).
x * (y \ (x * z)) = ((x * (y \ x)) / (x \ x)) * z.
((z * x) / y) * x = z * ((x / x) \ ((x / y) * x)).
end_of_list.
```

□

6.1.6 Quasigroup

O seguinte teorema (referência MKL4) foi fornecido pelo Professor Michael Kinyon (University of Denver). Pode ser consultado no artigo *A quasigroup with any of Moufang identities is a loop* da autoria de Scerbacov Victor and Izbas Vladimir [26].

Teorema 6.1.38 (MKL4)**.** *[26] Let Q be a quasigroup satisfying one of the Moufang identities. Then Q is a loop.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).

formulas(assumptions).
% quasigroup axioms
x \ (x * y) = y. x * (x \ y) = y.
(x * y) / y = x. (x / y) * y = x.
(x / y) \ x = y. x / (y \ x) = y.
% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.1.7 Demonic Refinement Algebra

Os seguintes teoremas (referências DRAnn) foram obtidos a partir do trabalho do Professor Peter Höfner (Australian National University) [21] . Foram selecionados os teoremas que originaram novos teoremas (secção 6.2) com provas de comprimento maior ou igual a 150 (`Max length proof >= 150`).

Teorema 6.1.39 (DRA22). *[21] Demonic Refinement Algebra DRA22; bubble sort w.r.t. star.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion laws
0*x = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%supremum property (proved)
(x+y)+z = z <-> (x+z=z & y+z=z).
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) + y = y -> x! * z + y = y.
(y*x + z) + y = y -> z * x! + y = y.
%isotonicity (proved)
x+y=y -> x! + y! = y!.
end_of_list.

formulas(goals).
% DRA22
%BubbleSort
y*x + x*y = x*y  ->  (x+y)! + x! *y! = x! *y!.
end_of_list.
```

□

Teorema 6.1.40 (DRA33). *[21] Demonic Refinement Algebra DRA33.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%
```

```
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion laws
0*x = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%supremum property (proved)
%    (x+y)+z = z <-> (x+z=z & y+z=z).
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) + y = y -> x! * z + y = y.
(y*x + z) + y = y -> z * x! + y = y.
%isotonicity (proved)
x+y=y -> x! + y! = y!.
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
y + (x*y + z) = x*y + z -> y + x@ *z= x@ *z.
%isotonicity (proved)
x+y=y -> x@ +y@ = y@.
%isolation
x@ = x! + x@ *0.
end_of_list.

formulas(goals).
% DRA33
(x+y)@*0 =0 -> x@*0+y@*0=0.
end_of_list.
```

□

Teorema 6.1.41 (DRA34). *[21] Demonic Refinement Algebra DRA34.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion laws
0*x = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%supremum property (proved)
%    (x+y)+z = z <-> (x+z=z & y+z=z).
% standard axioms for finite iteration (star) %%%%%%%%%
```

```
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) + y = y -> x! * z + y = y.
(y*x + z) + y = y -> z * x! + y = y.
%isotonicity (proved)
x+y=y -> x! + y! = y!.
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
y + (x*y + z) = x*y + z -> y + x@ *z= x@ *z.
%isotonicity (proved)
x+y=y -> x@ +y@ = y@.
%isolation
x@ = x! + x@ *0.
end_of_list.

formulas(goals).
% DRA34
y*x + x*(x+y)@= x*(x+y)@ -> ((x+y)@*0 =0 <- x@*0+y@*0=0).
end_of_list.
```

□

Teorema 6.1.42 (DRA37). *[21] Demonic Refinement Algebra DRA37.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion laws
0*x = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%inequality-axioms
x <= x.
x<=y & y<=z -> x<=z.
x<=y -> x+z<=y+z.
x<=y -> z+x<=z+y.
x<=y -> x*z<=y*z.
x<=y -> z*x<=z*y.
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) <= y -> x! * z <= y.
(y*x + z) <= y -> z * x! <= y.
%isotonicity (proved)
x<=y -> x! <= y!.
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
y <= x*y + z -> y <= x@ *z.
%isotonicity (proved)
x<=y -> x@ <= y@.
```

```
%isolation
x@ = x! + x@ *0.
%additional lemmas DRA17-19
x@*(y*x@)@ = (x+y)@.
x*(y*x)@ = (x*y)@*x.
end_of_list.

formulas(goals).
% DRA37
(x+y)@ <= (x!*y)@*x@.
(x!*y)@*x@ <= (x+y)@.
end_of_list.
```

□

6.1.8 Left Kleene Algebra

Os seguintes teoremas (referências LKAn) foram obtidos a partir do trabalho do Professor Peter Höfner (Australian National University) [21] . Foram selecionados os teoremas que originaram novos teoremas (secção 6.2) com provas de comprimento maior ou igual a 150 (`Max length proof >= 150`).

Teorema 6.1.43 (LKA1). *[21] Left Kleene Algebra LKA1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%left annihilation
0*x = 0.
%idempotence
x + x = x.
%left distributivity
(x + y)*z = x*z + y*z.
%subsumption order
x <= y <-> x+y=y.
%right superdistributivity
x*y + x*z <= x*(y + z).
% lazy Kleene algebras %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%finite iteration (star)
%left unfold
1 + x * x! <= x!.
%left induction
x*y + z <= y -> x! * z <= y.
end_of_list.

formulas(goals).
% LKA1
x <= x!.
end_of_list.
```

□

Teorema 6.1.44 (LKA7). *[21] Left Kleene Algebra LKA7; semi-selfcommutation I.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%left annihilation
0*x = 0.
%idempotence
x + x = x.
%left distributivity
(x + y)*z = x*z + y*z.
%subsumption order
x <= y <-> x+y=y.
%right superdistributivity
x*y + x*z <= x*(y + z).
% lazy Kleene algebras %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%finite iteration (star)
%left unfold
1 + x * x! <= x!.
%left induction
x*y + z <= y -> x! * z <= y.
end_of_list.

formulas(goals).
% LKA7
x!*x <= x*x!.
end_of_list.
```

□

6.1.9 Modal I-Semiring

Os seguintes teoremas (referências MISnn) foram obtidos a partir do trabalho do Professor Peter Höfner (Australian National University) [21] . Foram selecionados os teoremas que originaram novos teoremas (secção 6.2) com provas de comprimento maior ou igual a 150 (`Max length proof >= 150`).

Teorema 6.1.45 (MIS10). *[21] Modal I-Semiring MIS10; Galois connection II.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
```

```
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                        % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS10
all x all p all q (test(p) & test(q) -> (fd(x,p) + q = q -> bb(x,q) + p = bb(x,q))).
end_of_list.
```

□

Teorema 6.1.46 (MIS12)**.** *[21] Modal I-Semiring MIS12.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
```

```
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                        % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS12
all x all p all q (test(p) & test(q) -> p*fd(x,q)=fd(p*x,q)).
end_of_list.
```

□

Teorema 6.1.47 (MIS16). *[21] Modal I-Semiring MIS16; conjugation II.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
```

```
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                        % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS16
all x all p all q (test(p) & test(q) -> (p*fd(x,q)=0 <-> q*bd(x,p)=0)).
end_of_list.
```

□

Teorema 6.1.48 (MIS17)**.** *[21] Modal I-Semiring MIS17; diamond for tests.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
```

```
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS17
all p all q(test(p) & test(q) -> fd(p,q) = p*q).
end_of_list.
```

□

Teorema 6.1.49 (MIS18). *[21] Modal I-Semiring MIS18; idempotence of diamond for tests.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
```

```
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                        % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS18
all p all q(test(p) & test(q) -> fd(p,fb(p,q)) = fd(p,q)).
end_of_list.
```

□

Teorema 6.1.50 (MIS19). *[21] Modal I-Semiring MIS19; a cancellativity law.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
%    x + y = y + x.
%    x + 0 = x.
%    x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion laws
0*x = 0 & x*0 = 0.
%idempotency
%    x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).
```

```
%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS19
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.1.51 (MIS20). *[21] Modal I-Semiring MIS20; a cancellativity law.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
%    x + y = y + x.
%    x + 0 = x.
%    x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion laws
0*x = 0 & x*0 = 0.
%idempotency
%    x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
```

```
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS20
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

Teorema 6.1.52 (MIS24). *[21] Modal I-Semiring MIS24; diamond conjugation II.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
```

```
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS24
fd(x,d(y))*d(z) = 0 <- d(y)*bd(x,d(z)) = 0.
end_of_list.
```

□

Teorema 6.1.53 (MIS25). *[21] Modal I-Semiring MIS25; box conjugation I.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
```

```
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS25
fb(x,d(y)) + d(z) = 1 <- d(y) + bb(x,d(z)) = 1.
end_of_list.
```

□

Teorema 6.1.54 (MIS26). *[21] Modal I-Semiring MIS26; box conjugation II.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS26
fb(x,d(y)) + d(z) = 1 -> d(y) + bb(x,d(z)) = 1.
end_of_list.
```

□

Teorema 6.1.55 (MIS27). *[21] Modal I-Semiring MIS27; galois connection I.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS27
fd(x,d(y))+d(z) = d(z) <- d(y)+bb(x,d(z)) = bb(x,d(z)).
end_of_list.
```

□

Teorema 6.1.56 (MIS28). *[21] Modal I-Semiring MIS28; galois connection II.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").
```

```
formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS28
fd(x,d(y))+d(z) = d(z) -> d(y)+bb(x,d(z)) = bb(x,d(z)).
end_of_list.
```

□

Teorema 6.1.57 (MIS29). *[21] Modal I-Semiring MIS29; galois connection III.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
```

```
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS29
bd(x,d(y))+d(z) = d(z) <- d(y)+fb(x,d(z)) = fb(x,d(z)).
end_of_list.
```

□

Teorema 6.1.58 (MIS30). *[21] Modal I-Semiring MIS30; galois connection IV.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).
```

```
%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS30
bd(x,d(y))+d(z) = d(z) -> d(y)+fb(x,d(z)) = fb(x,d(z)).
end_of_list.
```

□

Teorema 6.1.59 (MIS31). *[21] Modal I-Semiring MIS31; Cancellation I.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
```

```
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS31
fd(x,bb(x,d(y)))+d(y)=d(y).
end_of_list.
```

□

Teorema 6.1.60 (MIS32). *[21] Modal I-Semiring MIS32; Cancellation II.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS32
```

```
d(y)+bb(x,fd(x,d(y))) = bb(x,fd(x,d(y))).
end_of_list.
```

□

Teorema 6.1.61 (MIS33). *[21] Modal I-Semiring MIS33; Cancellation III.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS33
bd(x,fb(x,d(y)))+d(y)=d(y).
end_of_list.
```

□

Teorema 6.1.62 (MIS34). *[21] Modal I-Semiring MIS34; Cancellation IV.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS34
d(y)+fb(x,bd(x,d(y))) = fb(x,bd(x,d(y))).
end_of_list.
```

□

Teorema 6.1.63 (MIS37). *[21] Modal I-Semiring MIS37; diamonds additivity.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
```

```
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS37
fd(x,d(y)+d(z)) = fd(x,d(y))+fd(x,d(z)).
end_of_list.
```

□

Teorema 6.1.64 (MIS38)**.** *[21] Modal I-Semiring MIS38; box multiplicativity.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
%semiring axioms
%commutative additive monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
%multiplicative monoid
x*(y*z) = (x*y)*z.
x*1 = x.
1*x = x.
%annihiltion laws
x*0 = 0.
0*x = 0.
%distributivty
x*(y+z) = x*y+x*z.
(x+y)*z = x*z+y*z.
%boolean domain axioms
a(x)*x = 0.
```

```
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% MIS38
fb(x,d(y)*d(z)) = fb(x,d(y))*fb(x,d(z)).
end_of_list.
```

□

Teorema 6.1.65 (MIS5). *[21] Modal I-Semiring MIS5; additivity II.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
```

```
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.

% proved lemata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dom(x+y)=dom(x)+dom(y).
end_of_list.

formulas(goals).
% MIS5
all x all p all q (test(p) & test(q) -> fd(x,p)+fd(x,q) = fd(x,p+q)).
end_of_list.
```

□

Teorema 6.1.66 (MIS6). *[21] Modal I-Semiring MIS6; covariance/contravariance.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
```

```
-test(y) -> fb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS6
all x all y all p (test(p) -> fb(x,fb(y,p)) = fb(x*y,p)).
end_of_list.
```

□

Teorema 6.1.67 (MIS8). *[21] Modal I-Semiring MIS8; multiplicativity.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS8
all x all p all q (test(p) & test(q) -> fb(x,p)*fb(x,q) = fb(x,p*q)).
end_of_list.
```

□

Teorema 6.1.68 (MIS9). *[21] Modal I-Semiring MIS9; Galois connection I.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
% standard axioms of i-semirings %%%%%%%%%%%%%%%%%%%%%%%
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x+(y+z) = (x+y)+z.
%multiplicative monoid
x*1 = x & 1*x = x.
x*(y*z) = (x*y)*z.
%annihiltion lawsy + x*y = x*y -> y + x^ = x^ .
0*x = 0 & x*0 = 0.
%idempotency
x + x = x.
%distributivty
x*(y + z) = x*y + x*z.
(x + y)*z = x*z + y*z.
%splitting lemma/supremum (IS3)
%  (x+y)+z=z <-> (x+z=z & y+z=z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                         % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% MIS9
all x all p all q (test(p) & test(q) -> (fd(x,p) + q = q <- bb(x,q) + p = bb(x,q))).
end_of_list.
```

□

6.1.10 Relation Algebra

Os seguintes teoremas (referências RAnn) foram obtidos a partir do trabalho do Professor Peter Höfner (Australian National University) [21] . Foram selecionados os teoremas que originaram novos teoremas (secção 6.2) com provas de comprimento maior ou igual a 150 (`Max length proof >= 150`).

Teorema 6.1.69 (RA14). *[21] Relation Algebra RA14.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% RA14
(x;y)*(x;z)' = x;(y*z')*(x;z)' -> (x;y)'+(x;z) = (x;(y*z'))'+x;z.
end_of_list.
```

□

Teorema 6.1.70 (RA17). *[21] Relation Algebra RA17.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
```

```
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% RA17
x;T=x -> (x*1);y=x*y.
end_of_list.
```

□

Teorema 6.1.71 (RA20). *[21] Relation Algebra RA20.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
end_of_list.

formulas(goals).
% RA20
x<=1 -> (x;T)*y=x;y.
end_of_list.
```

□

Teorema 6.1.72 (RA22). *[21] Relation Algebra RA22.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
end_of_list.

formulas(goals).
% RA22
x<=1 & y<=1 -> x;y=x*y.
end_of_list.
```

□

Teorema 6.1.73 (RA23). *[21] Relation Algebra RA23; distributivity.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
formulas(sos).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
```

```
0=x*x'.
end_of_list.

formulas(goals).
% RA23
x + 1 = 1 & y + 1 = 1 ->
(x ; z * y ; z) + (x * y) ; z = (x * y) ; z
& (x ; z * y ; z) + (x * y) ; z = x ; z * y ; z.
end_of_list.
```

□

Teorema 6.1.74 (RA24). *[21] Relation Algebra RA24.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% RA24
x+1=1 -> (x;y)*z' = (x;y)*(x;z)'.
end_of_list.
```

□

Teorema 6.1.75 (RA27). *[21] Relation Algebra RA27.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
```

```
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% proved lemmas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA27
x;T=x ->
(x*y);z+(x*(y;z))=x*(y;z)
&
(x*y);z+(x*(y;z))=(x*y);z.
end_of_list.
```

□

Teorema 6.1.76 (RA28). *[21] Relation Algebra RA28.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% RA28
x;T=x ->   (y*x@);(x*z)+y;(x*z)=y;(x*z)
& (y*x@);(x*z)+(y*x@);z=(y*x@);z
```

```
& y;(x*z)+(y*x@);(x*z)=(y*x@);(x*z)
& (y*x@);z+(y*x@);(x*z)=(y*x@);(x*z).
end_of_list.
```

□

Teorema 6.1.77 (RA38). *[21] Relation Algebra RA38.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

%bounded elements
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% RA38
(x;T);x =x <-> (x;x'@);x=0.
end_of_list.
```

□

Teorema 6.1.78 (RA40). *[21] Relation Algebra RA40.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).
```

```
% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

%greatest/smallest element
T = x+x'.
0 = x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
%row/column equivalence
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).

%Schroeder
x;y*z=0 <-> y*(x@;z)=0.
%Dedekind
(z;x*y) + (z*y;x@);(x*z@;y) = (z*y;x@);(x*z@;y) .
end_of_list.

formulas(goals).
% RA40
syq(x',y') = syq(x,y).
syq(x,y)@ = syq(y,x).
end_of_list.
```

□

Teorema 6.1.79 (RA42). *[21] Relation Algebra RA42.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.


% Schroeder %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;y+z'=z' <-> y'+x@;z=y'.
x@;z+y'=y' <-> y;z@+x'=x'.

% order props %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x+y=y <-> x@+y@=y@.
x+y=y <-> x'+y'=x'.
end_of_list.
```

```
formulas(goals).
% RA42
(x;y);x+x=x <-> y+((x;x'@);x)'@=((x;x'@);x)'@.
end_of_list.
```

□

Teorema 6.1.80 (RA47). *[21] Relation Algebra RA47.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA47
x;(y*z) = x;y * x;z -> x@;x<=1.
end_of_list.
```

□

Teorema 6.1.81 (RA48). *[21] Relation Algebra RA48; symmetric quotient is reflexive.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
```

```
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA48
1<= syq(x,x).
end_of_list.
```

□

Teorema 6.1.82 (RA49). *[21] Relation Algebra RA49; isotonicity of symmentric quotient.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).
end_of_list.

formulas(goals).
% RA49
syq(x,y) <= syq(z;x,z;y).
end_of_list.
```

□

Teorema 6.1.83 (RA50). *[21] Relation Algebra RA50; an unfold law.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA50
x<=(x;x@);x.
end_of_list.
```

□

Teorema 6.1.84 (RA51). *[21] Relation Algebra RA51; an important property for subidentities.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA51
x<=1 -> (x;T)' = (1*(x'));T.
end_of_list.
```

□

Teorema 6.1.85 (RA52). *[21] Relation Algebra RA52; a prop of symmentric quotient.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.
```

```
% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA52
x@;x<=1 & 1<=x;x@ -> x;syq(y,z) = syq(y;x@,z).
end_of_list.
```

□

Teorema 6.1.86 (RA53). *[21] Relation Algebra RA53.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA53
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.1.87 (RA54). *[21] Relation Algebra RA54; some strictness property.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA54
x!=0 <-> x@;x!=0.
end_of_list.
```

□

Teorema 6.1.88 (RA55). *[21] Relation Algebra RA55; complement of xT is left ideal.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA55
(x;T)' = (x;T)';T.
end_of_list.
```

□

Teorema 6.1.89 (RA56). *[21] Relation Algebra RA56; associativity of demonic composition.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA56
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

Teorema 6.1.90 (RA57). *[21] Relation Algebra RA57; consequense of the Tarski rule.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA57
(x;T);x = x -> (x;(x')@);x = 0.
end_of_list.
```

□

Teorema 6.1.91 (RA58). *[21] Relation Algebra RA58; meet splitting.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).
```

```
% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA58
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

Teorema 6.1.92 (RA59). *[21] Relation Algebra RA59; join splitting.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA59
x+y<=z <-> (x<=z & y<=z).
end_of_list.
```

□

Teorema 6.1.93 (RA60). *[21] Relation Algebra RA60; relational unfold with subidentity and converse.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% RA60
x = (1*x;x@);x.
end_of_list.
```

□

Teorema 6.1.94 (RA7). *[21] Relation Algebra RA7; Schroeder I.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
```

```
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% RA7
(x;y)*z=0 -> y*(x@;z)=0.
end_of_list.
```

□

Teorema 6.1.95 (RA8). *[21] Relation Algebra RA8; Schroeder II.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% RA8
(x;y)*z=0 <- y*(x@;z)=0.
end_of_list.
```

□

Teorema 6.1.96 (RA9). *[21] Relation Algebra RA9.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
% axioms of Boolean algebra (Maddux 2006) %%%%%%%%%%%%%%%%%%%%%%
x + y = y + x.
x+(y+z) = (x+y)+z.
x = (x' + y')'+(x'+y)'.
x*y = (x'+y')'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
end_of_list.

formulas(goals).
% RA9
(y;x)';x@ <= y'.
end_of_list.
```

□

6.1.11 Cancellation semigroup with a power-like map

O seguinte lema (referência PLM1) foi obtido a partir do artigo *Commutativity Theorems in Groups with Power-like Maps* da autoria de R. Padmanabhan and Yang Zhang [22].

Lema 6.1.12 (PLM1). *[22] Power-like maps, lemma 1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%% Cancellation semigroups
(x * y) * z = x * (y * z).
x * y != x * z | y = z.
x * y != z * y | x = z.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
```

```
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

6.1.12 Group with a power-like map

O seguinte lema (referência PLM1) foi obtido a partir do artigo *Commutativity Theorems in Groups with Power-like Maps* da autoria de R. Padmanabhan and Yang Zhang [22].

Teorema 6.1.97 (PLM2). *[22] Power-like maps, theorem 2.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
%% Group theory
(x * y) * z = x * (y * z).
x * e = x.
x * x' = e.
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

6.1.13 Hoop

Os seguintes teoremas e lemas foram obtidos a partir do artigo *Studying Algebraic Structures Using Prover9 and Mace4* da autoria de Rob Arthan and Paulo Oliva [23].

Teorema 6.1.98 (hp-sum-lemma). *[23] Theorem 1.1.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Reading from file hoop-eq.ax
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
(x * y) * z = x * (y + z).
x + (y * x) = y + (x * y).
x * 1 = 0.

% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
```

```
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.1.13 (MNA). *[23] Lemma 2.3.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
x * 0 = x.
z * (x + y) = (z * y) * x.
x + (y * x) = y + (x * y).
1 + x = 1.
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.1.14 (MNMN). *[23] Lemma 2.4.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
x * 0 = x.
z * (x + y) = (z * y) * x.
x + (y * x) = y + (x * y).
1 + x = 1.
% MNMN
x cup y = x + (y * x).
```

```
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.1.15 (NPJSSO). *[23] Lemma 2.5.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
x * 0 = x.
z * (x + y) = (z * y) * x.
x + (y * x) = y + (x * y).
1 + x = 1.
% NPJSSO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.1.16 (NSNSM). *[23] Lemma 2.6.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").
```

```
formulas(assumptions).
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
x * 0 = x.
z * (x + y) = (z * y) * x.
x + (y * x) = y + (x * y).
1 + x = 1.
% NSNSM
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = (x * y)' * (y cap x).
end_of_list.
```

□

Lema 6.1.17 (NNSSNN). *[23] Lemma 2.7.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(max_seconds, 600).
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
x * 0 = x.
z * (x + y) = (z * y) * x.
x + (y * x) = y + (x * y).
1 + x = 1.
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
```

```
end_of_list.
```

□

Lema 6.1.18 (SNNNO). *[23] Lemma 2.8.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(max_seconds, 600).
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
x * 0 = x.
z * (x + y) = (z * y) * x.
x + (y * x) = y + (x * y).
1 + x = 1.
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.1.19 (SSNNSNO). *[23] Lemma 2.9.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
x * 0 = x.
z * (x + y) = (z * y) * x.
x + (y * x) = y + (x * y).
1 + x = 1.
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
```

```
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

Teorema 6.1.99 (theoremNNSNNSNN-eq-basic-lemmas). *[23] Theorem 2.10*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Reading from file hoop-eq.ax
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x * x = 0.
0 * x = 0.
x * 0 = x.
z * (x + y) = (z * y) * x.
x + (y * x) = y + (x * y).
1 + x = 1.
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.1.14 Relational Lattice

Os seguintes teoremas (referências RL_n) foram obtidos a partir do artigo *Relational Lattices: From Databases to Universal Algebra* da autoria de Tadeusz Litaka, Szabolcs Mikulásb and Jan Hidders [27]

Teorema 6.1.100 (RL_1). *[27] Theorem 3.4, Clause 7.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x ^ y = y ^ x.
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.
x ^ (x v y) = x.
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.1.101 (RL_2). *[27] Theorem 3.4, Clause 8.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x ^ y = y ^ x.
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.
x ^ (x v y) = x.
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) = (((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) = UpMe(a,x2,z2) -> UpMe(x2,y2,z2) =
LoMe(x2,y2,z2))).
end_of_list.
```

□

Teorema 6.1.102 (RL_3). *[27] Theorem 3.4, Clause 9.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
x ^ y = y ^ x.
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.
x ^ (x v y) = x.
% RL_3
UpMe(x,y,z) = x ^ (y v z).
```

```
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) = (((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) = UpMe(a,x2,z2) -> (UpMe(a,x2,y2) =
UpMe(a,y2,z2) -> UpJo(x2,y2,z2) = LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.1.15 Hoare Triples

Os seguintes teoremas (referências HT_L5_2_X) foram obtidos a partir do artigo *Concurrent Kleene Algebra* da autoria de C. A. R. Tony Hoare, Bernhard Möller, Georg Struth and Ian Wehrman [28]

Teorema 6.1.103 (HT_L5_2_A)**.** *[28] Lemma 5.2, ordered monoid.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% partial order
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
% ordered monoid
x*(y*z) = (x*y)*z.
1*x = x.
x*1 = x.
x < y -> x*z < y*z.
y < z -> x*y < x*z.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.1.104 (HT_L5_2_B)**.** *[28] Lemma 5.2, ordered monoid.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% partial order
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
% ordered monoid
x*(y*z) = (x*y)*z.
1*x = x.
x*1 = x.
x < y -> x*z < y*z.
y < z -> x*y < x*z.
% HT_L5_2_B
% Hoare triple
```

```
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.1.105 (HT_L5_2_C). *[28] Lemma 5.2, ordered monoid.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% partial order
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
% ordered monoid
x*(y*z) = (x*y)*z.
1*x = x.
x*1 = x.
x < y -> x*z < y*z.
y < z -> x*y < x*z.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.1.106 (HT_L5_2_D). *[28] Lemma 5.2, ordered monoid.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% partial order
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
% ordered monoid
x*(y*z) = (x*y)*z.
1*x = x.
x*1 = x.
x < y -> x*z < y*z.
y < z -> x*y < x*z.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.1.107 (HT_L5_2_E). *[28] Lemma 5.2, ordered monoid.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% partial order
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
% ordered monoid
x*(y*z) = (x*y)*z.
1*x = x.
x*1 = x.
x < y -> x*z < y*z.
y < z -> x*y < x*z.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2 Teoremas gerados

Os seguintes teoremas, lemas e proposições foram obtidos a partir dos teoremas geradores apresentados na secção 6.1 e da base de dados de teorias do anexo 8.2. Os teoremas são apresentados na forma *Na teoria A o resultado B é verdadeiro*, em que A corresponde a uma dada teoria e B se refere a um determinado teorema. Os teoremas gerados com uma prova de comprimento maior ou igual a 150, tem a indicação `Max length proof` e respetivo comprimento.

6.2.1 Medial quasigroups

Teorema 6.2.1. *Na teoria Medial quasigroups o resultado 6.1.37 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% Medial quasigroups
(x * y) / y = x.
(x / y) * y = x.
x \ (x * y) = y.
x * (x \ y) = y.

(x * y) * (z * w) = (x * z) * (y * w).
% MKL3

end_of_list.
formulas(goals).
x * (y \ (x * z)) = ((x * (y \ x)) / (x \ x)) * z.
((z * x) / y) * x = z * ((x / x) \ ((x / y) * x)).
end_of_list.
```

□

Teorema 6.2.2. *Na teoria Medial quasigroups o resultado 6.1.38 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% Medial quasigroups
(x * y) / y = x.
(x / y) * y = x.
x \ (x * y) = y.
x * (x \ y) = y.

(x * y) * (z * w) = (x * z) * (y * w).
% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.2.2 Left Bol loop

Teorema 6.2.3. *Na teoria Left Bol loop o resultado 6.1.36 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
assign(eq_defs,fold).
formulas(assumptions).
% Left Bol loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
x * (y * (x * z)) = (x * (y * x)) * z.
% MKL1
% inner mappings
T(x,y) = y' * (x * y).
R(x,y,z) = ((x * y) * z) * (y * z)'.
L(x,y,z) = (z * y)' * (z * (y * x)).

% automorphic
T(x,y) * T(z,y) = T(x * z,y).
R(x,y,z) * R(u,y,z) = R(x * u,y,z).
L(x,y,z) * L(u,y,z) = L(x * u,y,z).
end_of_list.
formulas(goals).
((x * y) * x) * z = x * (y * (x * z)).
end_of_list.
```

□

Teorema 6.2.4. *Na teoria Left Bol loop o resultado 6.1.38 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% Left Bol loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
x * (y * (x * z)) = (x * (y * x)) * z.
% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.2.3 Right Bol loop

Teorema 6.2.5. *Na teoria Right Bol loop o resultado 6.1.38 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% Right Bol loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
z * ((x * y) * x) = ((z * x) * y) * x.
% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.2.4 Left conjugacy closed loop

Teorema 6.2.6. *Na teoria Left conjugacy closed loop o resultado 6.1.38 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% Left conjugacy closed loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
z * (y * x) = ((z * y) / z) * (z * x).
% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.2.5 Right conjugacy closed loop

Teorema 6.2.7. *Na teoria Right conjugacy closed loop o resultado 6.1.38 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% Right conjugacy closed loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
(x * y) * z = (x * z) * (z \ (y * z)).
% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.2.6 RIF loop

Teorema 6.2.8. *Na teoria RIF loop o resultado 6.1.38 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% RIF loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
(x * y) * (z * (x * y)) = ((x * (y * z)) * x) * y.
% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.2.7 ARIF loop

Teorema 6.2.9. *Na teoria ARIF loop o resultado 6.1.38 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% ARIF loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
(z * x) * ((y * x) * y) = (z * ((x * y) * x)) * y.
% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.2.8 Extra loop

Teorema 6.2.10. *Na teoria Extra loop o resultado 6.1.36 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
assign(eq_defs,fold).
formulas(assumptions).
% Extra loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
(x * (y * z)) * y = (x * y) * (z * y).

% MKL1
% inner mappings
T(x,y) = y' * (x * y).
R(x,y,z) = ((x * y) * z) * (y * z)'.
L(x,y,z) = (z * y)' * (z * (y * x)).

% automorphic
T(x,y) * T(z,y) = T(x * z,y).
R(x,y,z) * R(u,y,z) = R(x * u,y,z).
L(x,y,z) * L(u,y,z) = L(x * u,y,z).
end_of_list.
formulas(goals).
((x * y) * x) * z = x * (y * (x * z)).
end_of_list.
```

□

Teorema 6.2.11. *Na teoria Extra loop o resultado 6.1.38 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
assign(order, kbo).
formulas(assumptions).
% Extra loop
```

```
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
(x * (y * z)) * y = (x * y) * (z * y).

% MKL4
% Moufang identities
((x * y) * x) * z = x * (y * (x * z)).
%((x * y) * z) * y = x * (y * (z * y)).
%(x * y) * (z * x) = x * ((y * z) * x).
%(x * y) * (z * x) = (x * (y * z)) * x.
end_of_list.
formulas(goals).
x / x = y \ y.
end_of_list.
```

□

6.2.9 BCI-algebras

Teorema 6.2.12. *Na teoria BCI-algebras o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.13. *Na teoria BCI-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
```

```
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.14. *Na teoria BCI-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.15. *Na teoria BCI-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.16. *Na teoria BCI-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.17. *Na teoria BCI-algebras o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.18. *Na teoria BCI-algebras o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
```

```
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

padzha2019

Teorema 6.2.19. *Na teoria BCI-algebras o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.20. *Na teoria BCI-algebras o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.21. *Na teoria BCI-algebras o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.22. *Na teoria BCI-algebras o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.23. *Na teoria BCI-algebras o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_8
% regular semigroup
```

```
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.24. *Na teoria BCI-algebras o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.1. *Na teoria BCI-algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.2. *Na teoria BCI-algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.3. *Na teoria BCI-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.1. *Na teoria BCI-algebras o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.2. *Na teoria BCI-algebras o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Proposição 6.2.4. *Na teoria BCI-algebras o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.25. *Na teoria BCI-algebras o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.26. *Na teoria BCI-algebras o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.27. *Na teoria BCI-algebras o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.28. *Na teoria BCI-algebras o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.29. *Na teoria BCI-algebras o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.30. *Na teoria BCI-algebras o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.31. *Na teoria BCI-algebras o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.32. *Na teoria BCI-algebras o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.33. *Na teoria BCI-algebras o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_51
%miu is defined by
```

```
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.34. *Na teoria BCI-algebras o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.35. *Na teoria BCI-algebras o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.36. *Na teoria BCI-algebras o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
```

```
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.37. *Na teoria BCI-algebras o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.38. *Na teoria BCI-algebras o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.39. *Na teoria BCI-algebras o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
```

```
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Lema 6.2.3. *Na teoria BCI-algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Teorema 6.2.40. *Na teoria BCI-algebras o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.41. *Na teoria BCI-algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.42. *Na teoria BCI-algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.43. *Na teoria BCI-algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.44. *Na teoria BCI-algebras o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.45. *Na teoria BCI-algebras o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.4. *Na teoria BCI-algebras o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.5. *Na teoria BCI-algebras o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCI-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.10 BCK-algebras

Teorema 6.2.46. *Na teoria BCK-algebras o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.47. *Na teoria BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.48. *Na teoria BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.49. *Na teoria BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.50. *Na teoria BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
```

```
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.51. *Na teoria BCK-algebras o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.52. *Na teoria BCK-algebras o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.53. *Na teoria BCK-algebras o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.54. *Na teoria BCK-algebras o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.55. *Na teoria BCK-algebras o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_6
```

```
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.56. *Na teoria BCK-algebras o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.57. *Na teoria BCK-algebras o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
```

```
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.58. *Na teoria BCK-algebras o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.5. *Na teoria BCK-algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.6. *Na teoria BCK-algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.7. *Na teoria BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.6. *Na teoria BCK-algebras o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.7. *Na teoria BCK-algebras o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Proposição 6.2.8. *Na teoria BCK-algebras o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.59. *Na teoria BCK-algebras o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.60. *Na teoria BCK-algebras o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.61. *Na teoria BCK-algebras o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.62. *Na teoria BCK-algebras o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.63. *Na teoria BCK-algebras o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.64. *Na teoria BCK-algebras o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
```

```
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.65. *Na teoria BCK-algebras o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.66. *Na teoria BCK-algebras o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.67. *Na teoria BCK-algebras o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.68. *Na teoria BCK-algebras o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.69. *Na teoria BCK-algebras o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.70. *Na teoria BCK-algebras o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.71. *Na teoria BCK-algebras o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.72. *Na teoria BCK-algebras o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.73. *Na teoria BCK-algebras o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Lema 6.2.8. *Na teoria BCK-algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Teorema 6.2.74. *Na teoria BCK-algebras o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
```

```
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.75. *Na teoria BCK-algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.76. *Na teoria BCK-algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
```

```
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.77. *Na teoria BCK-algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.78. *Na teoria BCK-algebras o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
```

```
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.79. *Na teoria BCK-algebras o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.9. *Na teoria BCK-algebras o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.10. *Na teoria BCK-algebras o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.11 Commutative BCK-algebras

Teorema 6.2.80. *Na teoria Commutative BCK-algebras o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.81. *Na teoria Commutative BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
```

```
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.82. *Na teoria Commutative BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.83. *Na teoria Commutative BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
```

```
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.84. *Na teoria Commutative BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.85. *Na teoria Commutative BCK-algebras o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.86. *Na teoria Commutative BCK-algebras o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.87. *Na teoria Commutative BCK-algebras o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.88. *Na teoria Commutative BCK-algebras o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_6
% regular semigroup
```

```
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.89. *Na teoria Commutative BCK-algebras o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.90. *Na teoria Commutative BCK-algebras o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
```

```
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.91. *Na teoria Commutative BCK-algebras o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.92. *Na teoria Commutative BCK-algebras o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
```

```
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.9. *Na teoria Commutative BCK-algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.10. *Na teoria Commutative BCK-algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.11. *Na teoria Commutative BCK-algebras o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
```

```
x * (x * y) = y * (y * x).
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.12. *Na teoria Commutative BCK-algebras o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.13. *Na teoria Commutative BCK-algebras o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.14. *Na teoria Commutative BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.11. *Na teoria Commutative BCK-algebras o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.12. *Na teoria Commutative BCK-algebras o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
```

```
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Lema 6.2.13. *Na teoria Commutative BCK-algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.14. *Na teoria Commutative BCK-algebras o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.15. *Na teoria Commutative BCK-algebras o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_39
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * b') * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.16. *Na teoria Commutative BCK-algebras o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.93. *Na teoria Commutative BCK-algebras o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
```

```
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.94. *Na teoria Commutative BCK-algebras o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.95. *Na teoria Commutative BCK-algebras o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.96. *Na teoria Commutative BCK-algebras o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.97. *Na teoria Commutative BCK-algebras o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.98. *Na teoria Commutative BCK-algebras o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
```

```
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.99. *Na teoria Commutative BCK-algebras o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.100. *Na teoria Commutative BCK-algebras o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.101. *Na teoria Commutative BCK-algebras o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.102. *Na teoria Commutative BCK-algebras o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.103. *Na teoria Commutative BCK-algebras o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.104. *Na teoria Commutative BCK-algebras o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.105. *Na teoria Commutative BCK-algebras o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.106. *Na teoria Commutative BCK-algebras o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.107. *Na teoria Commutative BCK-algebras o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.108. *Na teoria Commutative BCK-algebras o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.17. *Na teoria Commutative BCK-algebras o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.15. *Na teoria Commutative BCK-algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.16. *Na teoria Commutative BCK-algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.109. *Na teoria Commutative BCK-algebras o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.110. *Na teoria Commutative BCK-algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.111. *Na teoria Commutative BCK-algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.112. *Na teoria Commutative BCK-algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.113. *Na teoria Commutative BCK-algebras o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.114. *Na teoria Commutative BCK-algebras o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
```

```
formulas(goals).
a' * a = f.
end_of_list.
```

□

Teorema 6.2.115. *Na teoria Commutative BCK-algebras o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.116. *Na teoria Commutative BCK-algebras o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.17. *Na teoria Commutative BCK-algebras o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.18. *Na teoria Commutative BCK-algebras o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative BCK-algebras
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
x * (x * y) = y * (y * x).
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.12 Pocrims

Lema 6.2.19. *Na teoria Pocrims o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Pocrims
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
```

```
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
(x * y) * z = x * (z @ y).
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.20. *Na teoria Pocrims o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Pocrims
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
(x * y) * z = x * (z @ y).
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.13 Directoids

Teorema 6.2.117. *Na teoria Directoids o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.118. *Na teoria Directoids o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.119. *Na teoria Directoids o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.120. *Na teoria Directoids o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
```

```
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.121. *Na teoria Directoids o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.122. *Na teoria Directoids o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.123. *Na teoria Directoids o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.124. *Na teoria Directoids o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.125. *Na teoria Directoids o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
```

```
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.126. *Na teoria Directoids o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.127. *Na teoria Directoids o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.128. *Na teoria Directoids o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.18. *Na teoria Directoids o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.19. *Na teoria Directoids o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.20. *Na teoria Directoids o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.21. *Na teoria Directoids o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.22. *Na teoria Directoids o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
```

```
end_of_list.
```

□

Lema 6.2.21. *Na teoria Directoids o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.22. *Na teoria Directoids o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Lema 6.2.23. *Na teoria Directoids o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
```

```
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.24. *Na teoria Directoids o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.25. *Na teoria Directoids o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.23. *Na teoria Directoids o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_34
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
```

```
R(a,b).
end_of_list.
```

□

Proposição 6.2.24. *Na teoria Directoids o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_40
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.25. *Na teoria Directoids o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.129. *Na teoria Directoids o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.130. *Na teoria Directoids o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.131. *Na teoria Directoids o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.132. *Na teoria Directoids o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.133. *Na teoria Directoids o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.134. *Na teoria Directoids o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.135. *Na teoria Directoids o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.136. *Na teoria Directoids o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.137. *Na teoria Directoids o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.138. *Na teoria Directoids o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.139. *Na teoria Directoids o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.140. *Na teoria Directoids o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.141. *Na teoria Directoids o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.142. *Na teoria Directoids o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.143. *Na teoria Directoids o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.144. *Na teoria Directoids o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.26. *Na teoria Directoids o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.26. *Na teoria Directoids o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.27. *Na teoria Directoids o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.28. *Na teoria Directoids o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.29. *Na teoria Directoids o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.27. *Na teoria Directoids o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.145. *Na teoria Directoids o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.146. *Na teoria Directoids o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
```

```
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.147. *Na teoria Directoids o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.148. *Na teoria Directoids o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
```

```
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.149. *Na teoria Directoids o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.150. *Na teoria Directoids o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
```

```
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.30. *Na teoria Directoids o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.31. *Na teoria Directoids o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
```

```
end_of_list.
```

□

Lema 6.2.32. *Na teoria Directoids o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Directoids
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.14 Normal bands

Teorema 6.2.151. *Na teoria Normal bands o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.152. *Na teoria Normal bands o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
```

```
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.153. *Na teoria Normal bands o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.154. *Na teoria Normal bands o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.155. *Na teoria Normal bands o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.156. *Na teoria Normal bands o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.157. *Na teoria Normal bands o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
```

```
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.158. *Na teoria Normal bands o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.159. *Na teoria Normal bands o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.160. *Na teoria Normal bands o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.161. *Na teoria Normal bands o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.162. *Na teoria Normal bands o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
```

```
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.163. *Na teoria Normal bands o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.28. *Na teoria Normal bands o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.29. *Na teoria Normal bands o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.30. *Na teoria Normal bands o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.31. *Na teoria Normal bands o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.32. *Na teoria Normal bands o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.33. *Na teoria Normal bands o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Proposição 6.2.34. *Na teoria Normal bands o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.33. *Na teoria Normal bands o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.34. *Na teoria Normal bands o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.35. *Na teoria Normal bands o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.35. *Na teoria Normal bands o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_40
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.36. *Na teoria Normal bands o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.164. *Na teoria Normal bands o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
```

```
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.165. *Na teoria Normal bands o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.166. *Na teoria Normal bands o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.167. *Na teoria Normal bands o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
```

```
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.168. *Na teoria Normal bands o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.169. *Na teoria Normal bands o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.170. *Na teoria Normal bands o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
```

```
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.171. *Na teoria Normal bands o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.172. *Na teoria Normal bands o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.173. *Na teoria Normal bands o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.174. *Na teoria Normal bands o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Proposição 6.2.37. *Na teoria Normal bands o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.36. *Na teoria Normal bands o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.37. *Na teoria Normal bands o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.38. *Na teoria Normal bands o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.39. *Na teoria Normal bands o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.38. *Na teoria Normal bands o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.175. *Na teoria Normal bands o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.176. *Na teoria Normal bands o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.177. *Na teoria Normal bands o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.178. *Na teoria Normal bands o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.179. *Na teoria Normal bands o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.180. *Na teoria Normal bands o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.40. *Na teoria Normal bands o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.41. *Na teoria Normal bands o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.42. *Na teoria Normal bands o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal bands
(x * y) * z = x * (y * z).
x * x = x.

((x * y) * z) * x = ((x * z) * y) * x.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.15 Order algebras

Teorema 6.2.181. *Na teoria Order algebras o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.182. *Na teoria Order algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.183. *Na teoria Order algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.184. *Na teoria Order algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
```

```
((x * y) * z) * y = (x * z) * y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.185. *Na teoria Order algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.186. *Na teoria Order algebras o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.187. *Na teoria Order algebras o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.188. *Na teoria Order algebras o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.189. *Na teoria Order algebras o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
```

```
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.190. *Na teoria Order algebras o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.191. *Na teoria Order algebras o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
```

```
end_of_list.
```

□

Teorema 6.2.192. *Na teoria Order algebras o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.193. *Na teoria Order algebras o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.39. *Na teoria Order algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.40. *Na teoria Order algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.41. *Na teoria Order algebras o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.42. *Na teoria Order algebras o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.43. *Na teoria Order algebras o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.44. *Na teoria Order algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.43. *Na teoria Order algebras o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.44. *Na teoria Order algebras o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.45. *Na teoria Order algebras o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
```

```
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.46. *Na teoria Order algebras o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.45. *Na teoria Order algebras o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_40
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.46. *Na teoria Order algebras o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
```

```
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.194. *Na teoria Order algebras o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.195. *Na teoria Order algebras o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.196. *Na teoria Order algebras o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.197. *Na teoria Order algebras o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.198. *Na teoria Order algebras o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.199. *Na teoria Order algebras o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.200. *Na teoria Order algebras o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.201. *Na teoria Order algebras o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_51
%miu is defined by
```

```
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.202. *Na teoria Order algebras o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.203. *Na teoria Order algebras o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.204. *Na teoria Order algebras o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
```

```
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.205. *Na teoria Order algebras o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.206. *Na teoria Order algebras o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Proposição 6.2.47. *Na teoria Order algebras o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
```

```
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.47. *Na teoria Order algebras o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.48. *Na teoria Order algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.49. *Na teoria Order algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_3
%relation L
```

```
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.50. *Na teoria Order algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.48. *Na teoria Order algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.207. *Na teoria Order algebras o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.208. *Na teoria Order algebras o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.209. *Na teoria Order algebras o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_9
```

```
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.210. *Na teoria Order algebras o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.211. *Na teoria Order algebras o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
```

```
((x * y) * z) * y = (x * z) * y.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.212. *Na teoria Order algebras o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.51. *Na teoria Order algebras o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").
```

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.52. *Na teoria Order algebras o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.53. *Na teoria Order algebras o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Order algebras
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
```

```
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.16 Semigroups with zero

Teorema 6.2.213. *Na teoria Semigroups with zero o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.214. *Na teoria Semigroups with zero o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.215. *Na teoria Semigroups with zero o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.216. *Na teoria Semigroups with zero o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.217. *Na teoria Semigroups with zero o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
```

```
end_of_list.
```

□

Teorema 6.2.218. *Na teoria Semigroups with zero o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.219. *Na teoria Semigroups with zero o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.220. *Na teoria Semigroups with zero o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
```

```
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.221. *Na teoria Semigroups with zero o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.222. *Na teoria Semigroups with zero o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.223. *Na teoria Semigroups with zero o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.224. *Na teoria Semigroups with zero o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.225. *Na teoria Semigroups with zero o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
```

```
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.49. *Na teoria Semigroups with zero o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.50. *Na teoria Semigroups with zero o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.51. *Na teoria Semigroups with zero o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
```

```
end_of_list.
```

□

Proposição 6.2.52. *Na teoria Semigroups with zero o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.53. *Na teoria Semigroups with zero o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.54. *Na teoria Semigroups with zero o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
```

```
end_of_list.
```

□

Lema 6.2.54. *Na teoria Semigroups with zero o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Proposição 6.2.55. *Na teoria Semigroups with zero o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.226. *Na teoria Semigroups with zero o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.227. *Na teoria Semigroups with zero o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.228. *Na teoria Semigroups with zero o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.229. *Na teoria Semigroups with zero o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.230. *Na teoria Semigroups with zero o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.231. *Na teoria Semigroups with zero o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.232. *Na teoria Semigroups with zero o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.233. *Na teoria Semigroups with zero o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.234. *Na teoria Semigroups with zero o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.235. *Na teoria Semigroups with zero o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.236. *Na teoria Semigroups with zero o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.237. *Na teoria Semigroups with zero o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.238. *Na teoria Semigroups with zero o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.239. *Na teoria Semigroups with zero o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.240. *Na teoria Semigroups with zero o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.241. *Na teoria Semigroups with zero o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
```

```
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.56. *Na teoria Semigroups with zero o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.55. *Na teoria Semigroups with zero o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.56. *Na teoria Semigroups with zero o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.57. *Na teoria Semigroups with zero o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.58. *Na teoria Semigroups with zero o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.57. *Na teoria Semigroups with zero o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.242. *Na teoria Semigroups with zero o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.243. *Na teoria Semigroups with zero o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
```

```
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.244. *Na teoria Semigroups with zero o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.245. *Na teoria Semigroups with zero o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
```

```
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.246. *Na teoria Semigroups with zero o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.247. *Na teoria Semigroups with zero o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.59. *Na teoria Semigroups with zero o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Semigroups with zero
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.17 Orthoimplication algebra

Teorema 6.2.248. *Na teoria Orthoimplication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.249. *Na teoria Orthoimplication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.250. *Na teoria Orthoimplication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.251. *Na teoria Orthoimplication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
```

```
end_of_list.
```

□

Teorema 6.2.252. *Na teoria Orthoimplication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.253. *Na teoria Orthoimplication algebra o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.254. *Na teoria Orthoimplication algebra o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
```

```
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.255. *Na teoria Orthoimplication algebra o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.256. *Na teoria Orthoimplication algebra o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.257. *Na teoria Orthoimplication algebra o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.258. *Na teoria Orthoimplication algebra o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.58. *Na teoria Orthoimplication algebra o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
```

```
end_of_list.
```

□

Proposição 6.2.59. *Na teoria Orthoimplication algebra o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.60. *Na teoria Orthoimplication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.61. *Na teoria Orthoimplication algebra o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.62. *Na teoria Orthoimplication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Lema 6.2.60. *Na teoria Orthoimplication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.61. *Na teoria Orthoimplication algebra o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
```

```
end_of_list.
```

□

Lema 6.2.62. *Na teoria Orthoimplication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.63. *Na teoria Orthoimplication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.64. *Na teoria Orthoimplication algebra o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.63. *Na teoria Orthoimplication algebra o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.2.64. *Na teoria Orthoimplication algebra o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.259. *Na teoria Orthoimplication algebra o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
```

```
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.260. *Na teoria Orthoimplication algebra o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.261. *Na teoria Orthoimplication algebra o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.262. *Na teoria Orthoimplication algebra o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
```

```
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.263. *Na teoria Orthoimplication algebra o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.264. *Na teoria Orthoimplication algebra o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.265. *Na teoria Orthoimplication algebra o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
```

```
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.266. *Na teoria Orthoimplication algebra o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.267. *Na teoria Orthoimplication algebra o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.268. *Na teoria Orthoimplication algebra o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_51
%miu is defined by
```

```
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.269. *Na teoria Orthoimplication algebra o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.270. *Na teoria Orthoimplication algebra o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.271. *Na teoria Orthoimplication algebra o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.272. *Na teoria Orthoimplication algebra o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.273. *Na teoria Orthoimplication algebra o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.274. *Na teoria Orthoimplication algebra o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.65. *Na teoria Orthoimplication algebra o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.65. *Na teoria Orthoimplication algebra o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.66. *Na teoria Orthoimplication algebra o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.67. *Na teoria Orthoimplication algebra o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.66. *Na teoria Orthoimplication algebra o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.275. *Na teoria Orthoimplication algebra o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
```

```
e = f.
end_of_list.
```

□

Teorema 6.2.276. *Na teoria Orthoimplication algebra o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.277. *Na teoria Orthoimplication algebra o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.278. *Na teoria Orthoimplication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.279. *Na teoria Orthoimplication algebra o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Teorema 6.2.280. *Na teoria Orthoimplication algebra o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.281. *Na teoria Orthoimplication algebra o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.68. *Na teoria Orthoimplication algebra o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Orthoimplication algebra
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
```

```
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.18 Orthomodular implication algebra

Teorema 6.2.282. *Na teoria Orthomodular implication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.283. *Na teoria Orthomodular implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
```

```
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.284. *Na teoria Orthomodular implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.285. *Na teoria Orthomodular implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.286. *Na teoria Orthomodular implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.287. *Na teoria Orthomodular implication algebra o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.288. *Na teoria Orthomodular implication algebra o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
```

```
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.289. *Na teoria Orthomodular implication algebra o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.290. *Na teoria Orthomodular implication algebra o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
```

```
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.291. *Na teoria Orthomodular implication algebra o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.292. *Na teoria Orthomodular implication algebra o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.67. *Na teoria Orthomodular implication algebra o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.68. *Na teoria Orthomodular implication algebra o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.69. *Na teoria Orthomodular implication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.70. *Na teoria Orthomodular implication algebra o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.71. *Na teoria Orthomodular implication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Lema 6.2.69. *Na teoria Orthomodular implication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_21
```

```
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.70. *Na teoria Orthomodular implication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.71. *Na teoria Orthomodular implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.72. *Na teoria Orthomodular implication algebra o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.72. *Na teoria Orthomodular implication algebra o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.2.73. *Na teoria Orthomodular implication algebra o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
```

```
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.293. *Na teoria Orthomodular implication algebra o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.294. *Na teoria Orthomodular implication algebra o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.295. *Na teoria Orthomodular implication algebra o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.296. *Na teoria Orthomodular implication algebra o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.297. *Na teoria Orthomodular implication algebra o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.298. *Na teoria Orthomodular implication algebra o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.299. *Na teoria Orthomodular implication algebra o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.300. *Na teoria Orthomodular implication algebra o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
```

```
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.301. *Na teoria Orthomodular implication algebra o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.302. *Na teoria Orthomodular implication algebra o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.303. *Na teoria Orthomodular implication algebra o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.304. *Na teoria Orthomodular implication algebra o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.305. *Na teoria Orthomodular implication algebra o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.306. *Na teoria Orthomodular implication algebra o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.307. *Na teoria Orthomodular implication algebra o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.308. *Na teoria Orthomodular implication algebra o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
```

```
end_of_list.
```

□

Proposição 6.2.74. *Na teoria Orthomodular implication algebra o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.73. *Na teoria Orthomodular implication algebra o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.74. *Na teoria Orthomodular implication algebra o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_3
```

```
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.75. *Na teoria Orthomodular implication algebra o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.75. *Na teoria Orthomodular implication algebra o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.309. *Na teoria Orthomodular implication algebra o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.310. *Na teoria Orthomodular implication algebra o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.311. *Na teoria Orthomodular implication algebra o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
((((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.312. *Na teoria Orthomodular implication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
((((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.313. *Na teoria Orthomodular implication algebra o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Teorema 6.2.314. *Na teoria Orthomodular implication algebra o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.315. *Na teoria Orthomodular implication algebra o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.76. *Na teoria Orthomodular implication algebra o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Orthomodular implication algebra
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.19 Quasi-implication algebra

Teorema 6.2.316. *Na teoria Quasi-implication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.317. *Na teoria Quasi-implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.318. *Na teoria Quasi-implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
```

```
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.319. *Na teoria Quasi-implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.320. *Na teoria Quasi-implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.321. *Na teoria Quasi-implication algebra o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.322. *Na teoria Quasi-implication algebra o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.323. *Na teoria Quasi-implication algebra o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.324. *Na teoria Quasi-implication algebra o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.325. *Na teoria Quasi-implication algebra o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.326. *Na teoria Quasi-implication algebra o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
```

```
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.76. *Na teoria Quasi-implication algebra o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.77. *Na teoria Quasi-implication algebra o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.78. *Na teoria Quasi-implication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.79. *Na teoria Quasi-implication algebra o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.80. *Na teoria Quasi-implication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Lema 6.2.77. *Na teoria Quasi-implication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.78. *Na teoria Quasi-implication algebra o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Lema 6.2.79. *Na teoria Quasi-implication algebra o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.80. *Na teoria Quasi-implication algebra o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.81. *Na teoria Quasi-implication algebra o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.81. *Na teoria Quasi-implication algebra o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.2.82. *Na teoria Quasi-implication algebra o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.327. *Na teoria Quasi-implication algebra o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.328. *Na teoria Quasi-implication algebra o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.329. *Na teoria Quasi-implication algebra o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.330. *Na teoria Quasi-implication algebra o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.331. *Na teoria Quasi-implication algebra o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.332. *Na teoria Quasi-implication algebra o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.333. *Na teoria Quasi-implication algebra o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.334. *Na teoria Quasi-implication algebra o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
```

```
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.335. *Na teoria Quasi-implication algebra o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.336. *Na teoria Quasi-implication algebra o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.337. *Na teoria Quasi-implication algebra o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.338. *Na teoria Quasi-implication algebra o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.339. *Na teoria Quasi-implication algebra o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.340. *Na teoria Quasi-implication algebra o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.341. *Na teoria Quasi-implication algebra o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.342. *Na teoria Quasi-implication algebra o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Lema 6.2.82. *Na teoria Quasi-implication algebra o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Proposição 6.2.83. *Na teoria Quasi-implication algebra o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.343. *Na teoria Quasi-implication algebra o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.344. *Na teoria Quasi-implication algebra o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
```

```
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.345. *Na teoria Quasi-implication algebra o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.346. *Na teoria Quasi-implication algebra o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
```

```
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.347. *Na teoria Quasi-implication algebra o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Teorema 6.2.348. *Na teoria Quasi-implication algebra o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
```

```
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.349. *Na teoria Quasi-implication algebra o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.83. *Na teoria Quasi-implication algebra o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Quasi-implication algebra
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
```

```
end_of_list.
```

□

6.2.20 Left closure semigroup

Teorema 6.2.350. *Na teoria Left closure semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.351. *Na teoria Left closure semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.352. *Na teoria Left closure semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.353. *Na teoria Left closure semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.354. *Na teoria Left closure semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
```

```
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.355. *Na teoria Left closure semigroup o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.356. *Na teoria Left closure semigroup o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.357. *Na teoria Left closure semigroup o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.358. *Na teoria Left closure semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.359. *Na teoria Left closure semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
```

```
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.360. *Na teoria Left closure semigroup o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.361. *Na teoria Left closure semigroup o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Proposição 6.2.84. *Na teoria Left closure semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.85. *Na teoria Left closure semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.86. *Na teoria Left closure semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.87. *Na teoria Left closure semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.88. *Na teoria Left closure semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.89. *Na teoria Left closure semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
```

```
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Lema 6.2.84. *Na teoria Left closure semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Proposição 6.2.90. *Na teoria Left closure semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.362. *Na teoria Left closure semigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.363. *Na teoria Left closure semigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.364. *Na teoria Left closure semigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.365. *Na teoria Left closure semigroup o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.366. *Na teoria Left closure semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.367. *Na teoria Left closure semigroup o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
```

```
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.368. *Na teoria Left closure semigroup o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.369. *Na teoria Left closure semigroup o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Proposição 6.2.91. *Na teoria Left closure semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
```

```
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.85. *Na teoria Left closure semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.86. *Na teoria Left closure semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.87. *Na teoria Left closure semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_3
```

```
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.88. *Na teoria Left closure semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.92. *Na teoria Left closure semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.370. *Na teoria Left closure semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.371. *Na teoria Left closure semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.372. *Na teoria Left closure semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_9
```

```
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.373. *Na teoria Left closure semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Lema 6.2.89. *Na teoria Left closure semigroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Left closure semigroup
(x * y) * z = x * (y * z).
```

```
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.21 Left closure monoid

Teorema 6.2.374. *Na teoria Left closure monoid o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.375. *Na teoria Left closure monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
```

```
d(x) * d(y) = d(y) * d(x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.376. *Na teoria Left closure monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.377. *Na teoria Left closure monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
```

```
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.378. *Na teoria Left closure monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.379. *Na teoria Left closure monoid o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.380. *Na teoria Left closure monoid o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.381. *Na teoria Left closure monoid o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.382. *Na teoria Left closure monoid o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_6
% regular semigroup
```

```
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.383. *Na teoria Left closure monoid o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.384. *Na teoria Left closure monoid o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
```

```
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.385. *Na teoria Left closure monoid o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Proposição 6.2.93. *Na teoria Left closure monoid o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.94. *Na teoria Left closure monoid o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.95. *Na teoria Left closure monoid o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.96. *Na teoria Left closure monoid o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
```

```
end_of_list.
```

□

Proposição 6.2.97. *Na teoria Left closure monoid o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.98. *Na teoria Left closure monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Lema 6.2.90. *Na teoria Left closure monoid o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
```

```
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Proposição 6.2.99. *Na teoria Left closure monoid o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.386. *Na teoria Left closure monoid o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
```

```
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.387. *Na teoria Left closure monoid o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.388. *Na teoria Left closure monoid o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.389. *Na teoria Left closure monoid o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.390. *Na teoria Left closure monoid o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.391. *Na teoria Left closure monoid o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
```

```
d(x) * d(y) = d(y) * d(x).
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.392. *Na teoria Left closure monoid o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.393. *Na teoria Left closure monoid o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Proposição 6.2.100. *Na teoria Left closure monoid o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.91. *Na teoria Left closure monoid o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.92. *Na teoria Left closure monoid o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
```

```
end_of_list.
```

□

Lema 6.2.93. *Na teoria Left closure monoid o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.94. *Na teoria Left closure monoid o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.101. *Na teoria Left closure monoid o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
```

```
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.394. *Na teoria Left closure monoid o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.395. *Na teoria Left closure monoid o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
```

```
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.396. *Na teoria Left closure monoid o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.397. *Na teoria Left closure monoid o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
```

```
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.398. *Na teoria Left closure monoid o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Lema 6.2.95. *Na teoria Left closure monoid o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Left closure monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
```

```
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.22 Inverse semigroup

Teorema 6.2.399. *Na teoria Inverse semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.400. *Na teoria Inverse semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
```

```
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.401. *Na teoria Inverse semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.402. *Na teoria Inverse semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.403. *Na teoria Inverse semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.404. *Na teoria Inverse semigroup o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.405. *Na teoria Inverse semigroup o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
```

```
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.406. *Na teoria Inverse semigroup o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.407. *Na teoria Inverse semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.408. *Na teoria Inverse semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.409. *Na teoria Inverse semigroup o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.410. *Na teoria Inverse semigroup o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
```

```
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.411. *Na teoria Inverse semigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.102. *Na teoria Inverse semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.103. *Na teoria Inverse semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.104. *Na teoria Inverse semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.105. *Na teoria Inverse semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.106. *Na teoria Inverse semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.107. *Na teoria Inverse semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Proposição 6.2.108. *Na teoria Inverse semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Proposição 6.2.109. *Na teoria Inverse semigroup o resultado 6.1.10 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_20
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
% inverse are unique
H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y)) -> H(y) = H(x').
end_of_list.
```

□

Lema 6.2.96. *Na teoria Inverse semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.97. *Na teoria Inverse semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Lema 6.2.98. *Na teoria Inverse semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.99. *Na teoria Inverse semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.100. *Na teoria Inverse semigroup o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.110. *Na teoria Inverse semigroup o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_27
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
a * a' = b * a'.
a * a' = a * b'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.2.111. *Na teoria Inverse semigroup o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_28
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = b * a'.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = a * b'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.2.112. *Na teoria Inverse semigroup o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_29
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = a * b'.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a' * a = b' * a.
```

```
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.2.113. *Na teoria Inverse semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_30
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = b' * a.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a * a' = a * b'.
a' * a = a' * b.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.2.114. *Na teoria Inverse semigroup o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_31
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = a' * b.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a * a' = a * b'.
a' * a = b' * a.
a = (a * b') * a.
a = (a * a') * b.
end_of_list.
```

□

Proposição 6.2.115. *Na teoria Inverse semigroup o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_33
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
end_of_list.
formulas(goals).
exists e (e * e = e & a = b * e).
a * a' = b * a'.
a * a' = a * b'.
a' * a = b' * a.
a' * a = a' * b.
a = (a * b') * a.
end_of_list.
```

□

Proposição 6.2.116. *Na teoria Inverse semigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_34
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.117. *Na teoria Inverse semigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_35
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = b * a'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.118. *Na teoria Inverse semigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_36
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = a * b'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.119. *Na teoria Inverse semigroup o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_37
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = b' * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.120. *Na teoria Inverse semigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_38
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = a' * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.121. *Na teoria Inverse semigroup o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_39
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * b') * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.122. *Na teoria Inverse semigroup o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_40
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.123. *Na teoria Inverse semigroup o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.2.124. *Na teoria Inverse semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.412. *Na teoria Inverse semigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.413. *Na teoria Inverse semigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
```

```
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.414. *Na teoria Inverse semigroup o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.415. *Na teoria Inverse semigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.416. *Na teoria Inverse semigroup o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
```

```
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.417. *Na teoria Inverse semigroup o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.418. *Na teoria Inverse semigroup o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.419. *Na teoria Inverse semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
```

```
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.420. *Na teoria Inverse semigroup o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.421. *Na teoria Inverse semigroup o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.422. *Na teoria Inverse semigroup o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.423. *Na teoria Inverse semigroup o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.424. *Na teoria Inverse semigroup o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.425. *Na teoria Inverse semigroup o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.426. *Na teoria Inverse semigroup o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.427. *Na teoria Inverse semigroup o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.125. *Na teoria Inverse semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.101. *Na teoria Inverse semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.102. *Na teoria Inverse semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.103. *Na teoria Inverse semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.104. *Na teoria Inverse semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.126. *Na teoria Inverse semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.428. *Na teoria Inverse semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.429. *Na teoria Inverse semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.430. *Na teoria Inverse semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_9
%relation L
```

```
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.431. *Na teoria Inverse semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.432. *Na teoria Inverse semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_11
%relation L
```

```
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.433. *Na teoria Inverse semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.105. *Na teoria Inverse semigroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
```

```
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.23 Domain range semigroup

Proposição 6.2.127. *Na teoria Domain range semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.128. *Na teoria Domain range semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
```

```
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Lema 6.2.106. *Na teoria Domain range semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Proposição 6.2.129. *Na teoria Domain range semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
```

```
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.434. *Na teoria Domain range semigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.435. *Na teoria Domain range semigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.436. *Na teoria Domain range semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.437. *Na teoria Domain range semigroup o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.438. *Na teoria Domain range semigroup o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.439. *Na teoria Domain range semigroup o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Lema 6.2.107. *Na teoria Domain range semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% AS2022_1
```

```
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.108. *Na teoria Domain range semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Domain range semigroup
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

6.2.24 Antidomain monoid

Teorema 6.2.440. *Na teoria Antidomain monoid o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.441. *Na teoria Antidomain monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.442. *Na teoria Antidomain monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.443. *Na teoria Antidomain monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.444. *Na teoria Antidomain monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.445. *Na teoria Antidomain monoid o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
```

```
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.446. *Na teoria Antidomain monoid o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.447. *Na teoria Antidomain monoid o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
```

```
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.448. *Na teoria Antidomain monoid o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.449. *Na teoria Antidomain monoid o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
```

```
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.450. *Na teoria Antidomain monoid o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.451. *Na teoria Antidomain monoid o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Proposição 6.2.130. *Na teoria Antidomain monoid o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.131. *Na teoria Antidomain monoid o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.132. *Na teoria Antidomain monoid o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
```

```
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.133. *Na teoria Antidomain monoid o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.134. *Na teoria Antidomain monoid o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
```

```
end_of_list.
```

□

Proposição 6.2.135. *Na teoria Antidomain monoid o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Lema 6.2.109. *Na teoria Antidomain monoid o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.110. *Na teoria Antidomain monoid o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Proposição 6.2.136. *Na teoria Antidomain monoid o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.452. *Na teoria Antidomain monoid o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
```

```
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.453. *Na teoria Antidomain monoid o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.454. *Na teoria Antidomain monoid o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
```

```
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.455. *Na teoria Antidomain monoid o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.456. *Na teoria Antidomain monoid o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.457. *Na teoria Antidomain monoid o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.458. *Na teoria Antidomain monoid o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.459. *Na teoria Antidomain monoid o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
```

```
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.460. *Na teoria Antidomain monoid o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.461. *Na teoria Antidomain monoid o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.462. *Na teoria Antidomain monoid o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.463. *Na teoria Antidomain monoid o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.464. *Na teoria Antidomain monoid o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_54
%miu is defined by
```

```
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.465. *Na teoria Antidomain monoid o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.466. *Na teoria Antidomain monoid o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.467. *Na teoria Antidomain monoid o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Lema 6.2.111. *Na teoria Antidomain monoid o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.112. *Na teoria Antidomain monoid o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
```

```
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.113. *Na teoria Antidomain monoid o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.114. *Na teoria Antidomain monoid o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.137. *Na teoria Antidomain monoid o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.468. *Na teoria Antidomain monoid o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.469. *Na teoria Antidomain monoid o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.470. *Na teoria Antidomain monoid o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.471. *Na teoria Antidomain monoid o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.472. *Na teoria Antidomain monoid o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
```

```
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.473. *Na teoria Antidomain monoid o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.115. *Na teoria Antidomain monoid o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% theoremNNSNNSNN-eq-basic-lemmas
```

```
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.116. *Na teoria Antidomain monoid o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.117. *Na teoria Antidomain monoid o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Antidomain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
```

```
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.25 Semilattice pseudo-complemented semigroup

Teorema 6.2.474. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.475. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.476. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.477. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.478. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_2
```

```
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.479. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.480. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.481. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.482. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.483. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
```

```
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.484. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.485. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
```

```
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Proposição 6.2.138. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.139. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.140. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.141. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.142. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.143. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Lema 6.2.118. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Proposição 6.2.144. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_41A
% H is a subsemigroup
```

```
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.486. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.487. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.488. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.489. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.490. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
```

```
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.491. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.492. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.493. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.494. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.495. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
```

```
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.496. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.497. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.498. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
```

```
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.499. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.500. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.501. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% IJ2022_57
```

```
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.145. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.119. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.120. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.121. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.122. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
```

```
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.146. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.502. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.503. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.504. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.505. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.506. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.507. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.123. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
```

```
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.124. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.125. *Na teoria Semilattice pseudo-complemented semigroup o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.26 Closable semilattice pseudo-complemented semigroup

Teorema 6.2.508. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.509. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.510. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
```

```
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.511. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.512. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.513. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.514. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.515. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.516. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.517. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_6
```

```
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.518. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.519. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
```

```
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Proposição 6.2.147. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.148. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.149. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.150. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.151. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
```

```
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.152. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Lema 6.2.126. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.127. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Proposição 6.2.153. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.520. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
```

```
x'' * x = x.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.521. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.522. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.523. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.524. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.525. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
```

```
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.526. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.527. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.528. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.529. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.530. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.531. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.532. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.533. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
```

```
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.534. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.535. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.154. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.128. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.129. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
```

```
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.130. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.131. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.155. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.536. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.537. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_8
%relation L
```

```
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.538. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.539. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
```

```
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.540. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.541. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.132. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.133. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.134. *Na teoria Closable semilattice pseudo-complemented semigroup o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Closable semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
x'' * x = x.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.27 Cancellative semigroup

Teorema 6.2.542. *Na teoria Cancellative semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
```

```
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.543. *Na teoria Cancellative semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.544. *Na teoria Cancellative semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.545. *Na teoria Cancellative semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.546. *Na teoria Cancellative semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.547. *Na teoria Cancellative semigroup o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.548. *Na teoria Cancellative semigroup o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.549. *Na teoria Cancellative semigroup o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.550. *Na teoria Cancellative semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
```

```
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.551. *Na teoria Cancellative semigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.552. *Na teoria Cancellative semigroup o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.553. *Na teoria Cancellative semigroup o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.554. *Na teoria Cancellative semigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.156. *Na teoria Cancellative semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
```

```
end_of_list.
```

□

Proposição 6.2.157. *Na teoria Cancellative semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.158. *Na teoria Cancellative semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.159. *Na teoria Cancellative semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.160. *Na teoria Cancellative semigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.161. *Na teoria Cancellative semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is associative
(H(x) + H(y)) + H(z) = H(x) + (H(y) + H(z)).
end_of_list.
```

□

Lema 6.2.135. *Na teoria Cancellative semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.136. *Na teoria Cancellative semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Lema 6.2.137. *Na teoria Cancellative semigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.138. *Na teoria Cancellative semigroup o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.162. *Na teoria Cancellative semigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_34
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.163. *Na teoria Cancellative semigroup o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.2.164. *Na teoria Cancellative semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
```

```
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.555. *Na teoria Cancellative semigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.556. *Na teoria Cancellative semigroup o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.557. *Na teoria Cancellative semigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
```

```
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.558. *Na teoria Cancellative semigroup o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.559. *Na teoria Cancellative semigroup o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.560. *Na teoria Cancellative semigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
```

```
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.561. *Na teoria Cancellative semigroup o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.562. *Na teoria Cancellative semigroup o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.563. *Na teoria Cancellative semigroup o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
```

```
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.564. *Na teoria Cancellative semigroup o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.565. *Na teoria Cancellative semigroup o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.566. *Na teoria Cancellative semigroup o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.567. *Na teoria Cancellative semigroup o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.568. *Na teoria Cancellative semigroup o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.165. *Na teoria Cancellative semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.139. *Na teoria Cancellative semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.140. *Na teoria Cancellative semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.141. *Na teoria Cancellative semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.142. *Na teoria Cancellative semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.166. *Na teoria Cancellative semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.569. *Na teoria Cancellative semigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.570. *Na teoria Cancellative semigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.571. *Na teoria Cancellative semigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.572. *Na teoria Cancellative semigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.573. *Na teoria Cancellative semigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.574. *Na teoria Cancellative semigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Lema 6.2.143. *Na teoria Cancellative semigroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Cancellative semigroup
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.28 Involutory quandle

Teorema 6.2.575. *Na teoria Involutory quandle o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.576. *Na teoria Involutory quandle o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.577. *Na teoria Involutory quandle o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
```

```
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.578. *Na teoria Involutory quandle o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.579. *Na teoria Involutory quandle o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.580. *Na teoria Involutory quandle o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.167. *Na teoria Involutory quandle o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.168. *Na teoria Involutory quandle o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.169. *Na teoria Involutory quandle o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.170. *Na teoria Involutory quandle o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.144. *Na teoria Involutory quandle o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.145. *Na teoria Involutory quandle o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Lema 6.2.146. *Na teoria Involutory quandle o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.147. *Na teoria Involutory quandle o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.171. *Na teoria Involutory quandle o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_34
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.172. *Na teoria Involutory quandle o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_35
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = b * a'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.173. *Na teoria Involutory quandle o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_36
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = a * b'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.174. *Na teoria Involutory quandle o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_37
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = b' * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.175. *Na teoria Involutory quandle o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_38
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = a' * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.176. *Na teoria Involutory quandle o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_39
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * b') * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.177. *Na teoria Involutory quandle o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_40
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.178. *Na teoria Involutory quandle o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.2.179. *Na teoria Involutory quandle o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.581. *Na teoria Involutory quandle o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.582. *Na teoria Involutory quandle o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.583. *Na teoria Involutory quandle o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.584. *Na teoria Involutory quandle o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.585. *Na teoria Involutory quandle o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.586. *Na teoria Involutory quandle o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.587. *Na teoria Involutory quandle o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Proposição 6.2.180. *Na teoria Involutory quandle o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.148. *Na teoria Involutory quandle o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.149. *Na teoria Involutory quandle o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.150. *Na teoria Involutory quandle o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Proposição 6.2.181. *Na teoria Involutory quandle o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.588. *Na teoria Involutory quandle o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.589. *Na teoria Involutory quandle o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.590. *Na teoria Involutory quandle o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% PLM1
```

```
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.151. *Na teoria Involutory quandle o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.152. *Na teoria Involutory quandle o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% PA2022_1
```

```
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.153. *Na teoria Involutory quandle o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Involutory quandle
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.29 Steiner quasigroup

Teorema 6.2.591. *Na teoria Steiner quasigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.592. *Na teoria Steiner quasigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.593. *Na teoria Steiner quasigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.594. *Na teoria Steiner quasigroup o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.595. *Na teoria Steiner quasigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.596. *Na teoria Steiner quasigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.597. *Na teoria Steiner quasigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_10
```

```
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.182. *Na teoria Steiner quasigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.183. *Na teoria Steiner quasigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.184. *Na teoria Steiner quasigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.185. *Na teoria Steiner quasigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.186. *Na teoria Steiner quasigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.154. *Na teoria Steiner quasigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.155. *Na teoria Steiner quasigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Lema 6.2.156. *Na teoria Steiner quasigroup o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.157. *Na teoria Steiner quasigroup o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.187. *Na teoria Steiner quasigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_34
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.188. *Na teoria Steiner quasigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_35
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = b * a'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.189. *Na teoria Steiner quasigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_36
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = a * b'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.190. *Na teoria Steiner quasigroup o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_37
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = b' * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.191. *Na teoria Steiner quasigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_38
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = a' * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.192. *Na teoria Steiner quasigroup o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_39
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * b') * a.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.193. *Na teoria Steiner quasigroup o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_40
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a = (a * a') * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.194. *Na teoria Steiner quasigroup o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.2.195. *Na teoria Steiner quasigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.598. *Na teoria Steiner quasigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.599. *Na teoria Steiner quasigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.600. *Na teoria Steiner quasigroup o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.601. *Na teoria Steiner quasigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.602. *Na teoria Steiner quasigroup o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.603. *Na teoria Steiner quasigroup o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.604. *Na teoria Steiner quasigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.605. *Na teoria Steiner quasigroup o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.606. *Na teoria Steiner quasigroup o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.607. *Na teoria Steiner quasigroup o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.608. *Na teoria Steiner quasigroup o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Proposição 6.2.196. *Na teoria Steiner quasigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.158. *Na teoria Steiner quasigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.159. *Na teoria Steiner quasigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.160. *Na teoria Steiner quasigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.161. *Na teoria Steiner quasigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.197. *Na teoria Steiner quasigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.609. *Na teoria Steiner quasigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.610. *Na teoria Steiner quasigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.611. *Na teoria Steiner quasigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
```

```
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.612. *Na teoria Steiner quasigroup o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.613. *Na teoria Steiner quasigroup o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
```

```
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.162. *Na teoria Steiner quasigroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.163. *Na teoria Steiner quasigroup o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.164. *Na teoria Steiner quasigroup o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner quasigroup
x * x = x.
x * y = y * x.
x * (x * y) = y.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.30 Steiner loop

Teorema 6.2.614. *Na teoria Steiner loop o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_1
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
```

□

Teorema 6.2.615. *Na teoria Steiner loop o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.616. *Na teoria Steiner loop o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.617. *Na teoria Steiner loop o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.618. *Na teoria Steiner loop o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
```

```
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.619. *Na teoria Steiner loop o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_3
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.620. *Na teoria Steiner loop o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_4
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.621. *Na teoria Steiner loop o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.622. *Na teoria Steiner loop o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.623. *Na teoria Steiner loop o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
```

```
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.624. *Na teoria Steiner loop o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_7
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.625. *Na teoria Steiner loop o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_8
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
```

□

Teorema 6.2.626. *Na teoria Steiner loop o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.198. *Na teoria Steiner loop o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.199. *Na teoria Steiner loop o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.200. *Na teoria Steiner loop o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_15
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(a * b)' = b' * a'.
end_of_list.
```

□

Proposição 6.2.201. *Na teoria Steiner loop o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.202. *Na teoria Steiner loop o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_18
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
L(x,y) <-> x' * x = y' * y.
end_of_list.
```

□

Proposição 6.2.203. *Na teoria Steiner loop o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.165. *Na teoria Steiner loop o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.166. *Na teoria Steiner loop o resultado 6.1.7 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_23
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is reflexive
R(x,x).
end_of_list.
```

□

Lema 6.2.167. *Na teoria Steiner loop o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_24
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is antisymmetric
R(x,y) & R(y,x) -> x = y.
end_of_list.
```

□

Lema 6.2.168. *Na teoria Steiner loop o resultado 6.1.9 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_25
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
end_of_list.
formulas(goals).
% R is transitive
R(x,y) & R(y,z) -> R(x,z).
end_of_list.
```

□

Proposição 6.2.204. *Na teoria Steiner loop o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_34
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.205. *Na teoria Steiner loop o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_35
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a * a' = b * a'.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.206. *Na teoria Steiner loop o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_38
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
a' * a = a' * b.
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.207. *Na teoria Steiner loop o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_41
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
O(a,b).
end_of_list.
formulas(goals).
O(x * a, x * b).
O(a * x, b * x).
O(a', b').
end_of_list.
```

□

Proposição 6.2.208. *Na teoria Steiner loop o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.627. *Na teoria Steiner loop o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.628. *Na teoria Steiner loop o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.629. *Na teoria Steiner loop o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.630. *Na teoria Steiner loop o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.631. *Na teoria Steiner loop o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.632. *Na teoria Steiner loop o resultado 6.1.25 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_47
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> sigma(z * x, x).
end_of_list.
```

□

Teorema 6.2.633. *Na teoria Steiner loop o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.634. *Na teoria Steiner loop o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
```

```
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.635. *Na teoria Steiner loop o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_50
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.636. *Na teoria Steiner loop o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.637. *Na teoria Steiner loop o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.638. *Na teoria Steiner loop o resultado 6.1.31 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_53
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.639. *Na teoria Steiner loop o resultado 6.1.32 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_54
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.640. *Na teoria Steiner loop o resultado 6.1.33 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_55
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> miu(z * x, x).
end_of_list.
```

□

Teorema 6.2.641. *Na teoria Steiner loop o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_56
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Teorema 6.2.642. *Na teoria Steiner loop o resultado 6.1.35 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% IJ2022_57
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
R(a,b).
end_of_list.
formulas(goals).
miu(a,b).
end_of_list.
```

□

Proposição 6.2.209. *Na teoria Steiner loop o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.169. *Na teoria Steiner loop o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.170. *Na teoria Steiner loop o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.171. *Na teoria Steiner loop o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.172. *Na teoria Steiner loop o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.210. *Na teoria Steiner loop o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.643. *Na teoria Steiner loop o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.644. *Na teoria Steiner loop o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.645. *Na teoria Steiner loop o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_9
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
L(x,z) & R(z,a) -> exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
end_of_list.
formulas(goals).
exists y (a * y = e & y * a = f).
end_of_list.
```

□

Teorema 6.2.646. *Na teoria Steiner loop o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
```

```
x * (x * y) = y.
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.647. *Na teoria Steiner loop o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% AS2022_11
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a * a' = e.
end_of_list.
```

□

Teorema 6.2.648. *Na teoria Steiner loop o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
```

```
% AS2022_12
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
exists u (L(a, u) & R(u, b)).
R(e,a).
L(e,b).
L(f,a).
R(f,b).
(a * a') * a = a.
(a' * a) * a' = a'.
L(a', b).%a' in Hb
R(a', b).
end_of_list.
formulas(goals).
a' * a = f.
end_of_list.
```

□

Teorema 6.2.649. *Na teoria Steiner loop o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.650. *Na teoria Steiner loop o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
```

```
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.173. *Na teoria Steiner loop o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.174. *Na teoria Steiner loop o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.175. *Na teoria Steiner loop o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Steiner loop
x * x = 1.
x * y = y * x.
x * (x * y) = y.
% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.31 Double Ward quasigroup

Teorema 6.2.651. *Na teoria Double Ward quasigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_2
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.652. *Na teoria Double Ward quasigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_2
```

```
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.653. *Na teoria Double Ward quasigroup o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_5
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Teorema 6.2.654. *Na teoria Double Ward quasigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
```

```
end_of_list.
```

□

Teorema 6.2.655. *Na teoria Double Ward quasigroup o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_6
% regular semigroup
(x * x') * x = x.
(x' * x) * x' = x'.
% idempotent elements commute
(x * x) = x & y * y = y -> x * y = y * x.
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
end_of_list.
```

□

Teorema 6.2.656. *Na teoria Double Ward quasigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_10
% R-relation
R(x,y) <-> x = y | exists u (exists v (x = y * u & y = x * v)).
% L-relation
L(x,y) <-> x = y | exists u (exists v (x = u * y & y = v * x)).
% each R-class has an idempotent
all x (exists y (R(x,y) & y * y = y)).
% each L-class has an idempotent
all x (exists y (L(x,y) & y * y = y)).
% in each R-class, there is at most one idempotent.
R(x,y) & x * x = x & y * y = y -> x = y.
% in each L-class, there is at most one idempotent.
L(x,y) & x * x = x & y * y = y -> x = y.
end_of_list.
formulas(goals).
% inverse semigroup (x' is the only inverse of x)
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
```

□

Proposição 6.2.211. *Na teoria Double Ward quasigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_13
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
(x')' = x.
end_of_list.
```

□

Proposição 6.2.212. *Na teoria Double Ward quasigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_14
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e -> e' = e.
end_of_list.
```

□

Proposição 6.2.213. *Na teoria Double Ward quasigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_16
% inverse semigroup (x' is the only inverse of x)
x = (x * x') * x. x' = (x' * x) * x'.
(x * y) * x = x & (y * x) * y = y & (x * z) * x = x & (z * x) * z = z -> y = z.
end_of_list.
formulas(goals).
e * e = e ->
((x * e) * x') * ((x * e) * x') = (x * e) * x'
```

```
&
((x' * e) * x) * ((x' * e) * x) = (x' * e) * x.
end_of_list.
```

□

Proposição 6.2.214. *Na teoria Double Ward quasigroup o resultado 6.1.8 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_19
%H:S->T is an homomorphism
H(x * y) = H(x) + H(y).
end_of_list.
formulas(goals).
%H(x) is regular
all x exists y (H(x) = H(x) + (H(y) + H(x)) & H(y) = H(y) + (H(x) + H(y))  ).
end_of_list.
```

□

Lema 6.2.176. *Na teoria Double Ward quasigroup o resultado 6.1.5 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_21
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
formulas(goals).
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
```

□

Lema 6.2.177. *Na teoria Double Ward quasigroup o resultado 6.1.6 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
```

```
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_22
% e and f are idempotents
e * e = e. f * f = f.
% c belongs to Sef
exists d (c = (d * e) * f).
end_of_list.
formulas(goals).
% c belongs to Se and to Sf
exists a (c = a * e).
exists b (c = b * f).
end_of_list.
```

□

Proposição 6.2.215. *Na teoria Double Ward quasigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_34
% relation R (natural order relation on an inverse semigroup)
R(x,y) <-> exists e (e * e = e & x = e * y).
exists e (e * e = e & a = b * e).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Proposição 6.2.216. *Na teoria Double Ward quasigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_41A
% H is a subsemigroup
H(a). % H is non-empty
H(x) & H(y) -> H(x * y). % H is a subsemigroup
H(x) -> H(x'). % H is closed under inversion
% relation O (natural order relation on an inverse semigroup)
O(x,y) <-> exists e (e * e = e & x = e * y).
% defining closure of H
Hw(y) <-> O(x,y) & H(x).
Kw(y) <-> O(x,y) & K(x).
H(x) -> Hw(x).
(H(x) -> K(x)) -> (Hw(x) -> Kw(x)).
% Hww = Hw
Hw(x) & O(x,y) -> Hw(y).
H(x) & H(y) -> H(x * y).
H(x) -> H(x').
```

```
end_of_list.
formulas(goals).
Hw(x) & Hw(y) -> Hw(x * y).
end_of_list.
```

□

Teorema 6.2.657. *Na teoria Double Ward quasigroup o resultado 6.1.20 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_42
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.658. *Na teoria Double Ward quasigroup o resultado 6.1.21 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_43
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.659. *Na teoria Double Ward quasigroup o resultado 6.1.22 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
```

```
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_44
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) & sigma(y,z) -> sigma(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.660. *Na teoria Double Ward quasigroup o resultado 6.1.23 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_45
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(x * z, y * z).
end_of_list.
```

□

Teorema 6.2.661. *Na teoria Double Ward quasigroup o resultado 6.1.24 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_46
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
sigma(x,y) -> sigma(z * x, z * y).
end_of_list.
```

□

Teorema 6.2.662. *Na teoria Double Ward quasigroup o resultado 6.1.26 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_48
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(sigma(x * y, z))).
end_of_list.
```

□

Teorema 6.2.663. *Na teoria Double Ward quasigroup o resultado 6.1.27 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_49
%E is a semillatice (commutative semigroup of idempotents)
E(x) -> x * x = x.
%sigma is defined by
sigma(x,y) <-> exists z(z * z = z & z * x = z * y).
%R is a group congruence
R(x,x).
R(x,y) -> R(y,x).
R(x,y) & R(y,z) -> R(x,z).
R(x,y) -> R(x * z, y * z).
R(x,y) -> R(z * x, z * y).
z * z = z -> R(z * x, x).
z * z = z -> all x(exists y(R(x * y, z))).
sigma(a,b).
end_of_list.
formulas(goals).
R(a,b).
end_of_list.
```

□

Teorema 6.2.664. *Na teoria Double Ward quasigroup o resultado 6.1.28 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_50
%miu is defined by
```

```
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,x). %reflexive
end_of_list.
```

□

Teorema 6.2.665. *Na teoria Double Ward quasigroup o resultado 6.1.29 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_51
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) -> miu(y,x). %symmetric
end_of_list.
```

□

Teorema 6.2.666. *Na teoria Double Ward quasigroup o resultado 6.1.30 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_52
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
miu(x,y) & miu(y,z) -> miu(x,z). %transitive
end_of_list.
```

□

Teorema 6.2.667. *Na teoria Double Ward quasigroup o resultado 6.1.34 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% IJ2022_56
```

```
%miu is defined by
miu(x,y) <-> (z * z = z -> (x' * z) * x = (y' * z) * y).
end_of_list.
formulas(goals).
z * z = z -> all x(exists y(miu(x * y, z))).
end_of_list.
```

□

Proposição 6.2.217. *Na teoria Double Ward quasigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% AS2022_0
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
R(x,y) & L(y,z) -> exists u (L(x,u) & R(u,z)).
end_of_list.
```

□

Lema 6.2.178. *Na teoria Double Ward quasigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% AS2022_1
%left ideal of y (x belongs to Sy)
LI(x,y) <-> x = y | exists z (z * y = x).
%relation R
L(y1,y2) <-> (LI(x,y1) <-> LI(x,y2)).
end_of_list.
formulas(goals).
L(w,z) <-> exists x (x * w = z) & exists y (y * z = w).
end_of_list.
```

□

Lema 6.2.179. *Na teoria Double Ward quasigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% AS2022_2
%right ideal of y (x belongs to yS)
RI(x,y) <-> x = y | exists z (y * z = x).
%relation R
R(y1,y2) <-> (RI(x,y1) <-> RI(x,y2)).
end_of_list.
formulas(goals).
R(w,z) <-> exists x (w * x = z) & exists y (z * y = w).
end_of_list.
```

□

Lema 6.2.180. *Na teoria Double Ward quasigroup o resultado 6.1.3 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% AS2022_3
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
end_of_list.
formulas(goals).
%right congruence
L(x,y) -> L(x*z, y*z).
end_of_list.
```

□

Lema 6.2.181. *Na teoria Double Ward quasigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% AS2022_4
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
end_of_list.
formulas(goals).
%left congruence
R(x,y) -> R(z * x, z * y).
end_of_list.
```

□

Proposição 6.2.218. *Na teoria Double Ward quasigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% AS2022_5
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
% a is regular
exists x ( (a * x) * a) = a.
exists u (L(a,u) & R(u,b)).
end_of_list.
formulas(goals).
exists y ((b * y) *b) =b.
end_of_list.
```

□

Teorema 6.2.668. *Na teoria Double Ward quasigroup o resultado 6.1.1 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% AS2022_6
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * b) * a = a. %regular
(a * a')* a = a. % inverse
(a' * a) *a' = a'. %inverse
end_of_list.
formulas(goals).
% a and a' in the same D class
exists u (L(a,u) & R(u,a')).
end_of_list.
```

□

Teorema 6.2.669. *Na teoria Double Ward quasigroup o resultado 6.1.2 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
```

```
x * y = e * ((e * y) * (e * x)).
% AS2022_8
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
(a * r) * a = a. %regular
R(e,a).
L(e,a).
L(f,a).
R(f,a).
e * (a * e) = e.
a * (e * a) = a.
f * (a * f) = f.
a * (f * a) = a.
end_of_list.
formulas(goals).
e = f.
end_of_list.
```

□

Teorema 6.2.670. *Na teoria Double Ward quasigroup o resultado 6.1.4 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% AS2022_10
%relation L
L(x,y) <-> (x = y | exists u exists v (x = u * y & y = v * x)).
%relation R
R(x,y) <-> (x = y | exists u exists v (x = y * u & y = x * v)).
e*e = e.
f*f = f.
Hb(x) <-> L(x,b) & R(x,b).
L(a, u). % b belongs D
R(u, b).
exists y (x * (y * x) = x).
R(e,a).
L(e,b).
R(f,b).
L(f,a).
%Hb(a').
a * a' = e.
a' * a = f.
end_of_list.
formulas(goals).
exists y (Hb(y) & a * (y * a) =a & y * (a * y) = y).
end_of_list.
```

□

Teorema 6.2.671. *Na teoria Double Ward quasigroup o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
```

```
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Lema 6.2.182. *Na teoria Double Ward quasigroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Double Ward quasigroup
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.32 Associative dialgebra

Teorema 6.2.672 (T_042_285)**.** *Na teoria Associative dialgebra o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 276):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Associative dialgebra
(x + y) + z = x + (y + z).
(x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z).
(x * y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.673 (T_042_287). *Na teoria Associative dialgebra o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 272):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Associative dialgebra
(x + y) + z = x + (y + z).
(x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z).
(x * y) * z = x * (y * z).
```

```
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

Teorema 6.2.674 (T_042_394)**.** *Na teoria Associative dialgebra o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 542):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Associative dialgebra
(x + y) + z = x + (y + z).
(x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z).
(x * y) * z = x * (y * z).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
```

```
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

6.2.33 Associative dimonoid

Teorema 6.2.675 (T_043_285). *Na teoria Associative dimonoid o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 278):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Associative dimonoid
x + (y + z) = (x + y) + z & (x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z) & x * (y * z) = (x * y) * z.

%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%%
```

```
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.676 (T_043_287). *Na teoria Associative dimonoid o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 274):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Associative dimonoid
x + (y + z) = (x + y) + z & (x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z) & x * (y * z) = (x * y) * z.

%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
```

```
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

Teorema 6.2.677 (T_043_394). *Na teoria Associative dimonoid o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 544):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Associative dimonoid
x + (y + z) = (x + y) + z & (x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z) & x * (y * z) = (x * y) * z.

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

6.2.34 Bounded lattice

Teorema 6.2.678 (T_001_348). *Na teoria Bounded lattice o resultado 6.1.69 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 175):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
```

```
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA14)
(x;y)*(x;z)' = x;(y*z')*(x;z)' -> (x;y)'+(x;z) = (x;(y*z'))'+x;z.
end_of_list.
```

□

Teorema 6.2.679 (T_001_358)**.** *Na teoria Bounded lattice o resultado 6.1.73 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 152):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
```

```
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA23)
x + 1 = 1 & y + 1 = 1 ->
(x ; z * y ; z) + (x * y) ; z = (x * y) ; z
& (x ; z * y ; z) + (x * y) ; z = x ; z * y ; z.
end_of_list.
```

□

Teorema 6.2.680 (T_001_359)**.** *Na teoria Bounded lattice o resultado 6.1.74 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 156):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA24)
x+1=1 -> (x;y)*z' = (x;y)*(x;z)'.
end_of_list.
```

□

Teorema 6.2.681 (T_001_363)**.** *Na teoria Bounded lattice o resultado 6.1.76 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 155):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA28)
x;T=x ->   (y*x@);(x*z)+y;(x*z)=y;(x*z)
& (y*x@);(x*z)+(y*x@);z=(y*x@);z
& y;(x*z)+(y*x@);(x*z)=(y*x@);(x*z)
& (y*x@);z+(y*x@);(x*z)=(y*x@);(x*z).
end_of_list.
```

□

Teorema 6.2.682 (T_001_386)**.** *Na teoria Bounded lattice o resultado 6.1.82 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 155):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
```

```
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).
end_of_list.

formulas(goals).
% Relation Algebra (RA49)
syq(x,y) <= syq(z;x,z;y).
end_of_list.
```

□

Teorema 6.2.683 (T_001_391). *Na teoria Bounded lattice o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 177):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
```

```
%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.684 (T_001_394)**.** *Na teoria Bounded lattice o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 166):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

Teorema 6.2.685 (T_001_396)**.** *Na teoria Bounded lattice o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 197):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

Teorema 6.2.686 (T_001_400)**.** *Na teoria Bounded lattice o resultado 6.1.94 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 160):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
```

```
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA7)
(x;y)*z=0 -> y*(x@;z)=0.
end_of_list.
```

□

Teorema 6.2.687 (T_001_401)**.** *Na teoria Bounded lattice o resultado 6.1.95 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 163):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Bounded lattice
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA8)
(x;y)*z=0 <- y*(x@;z)=0.
end_of_list.
```

□

6.2.35 Commutative dimonoid

Teorema 6.2.688 (T_039_062)**.** *Na teoria Commutative dimonoid o resultado 6.1.39 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 214):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) + y = y -> x! * z + y = y.
(y*x + z) + y = y -> z * x! + y = y.
%isotonicity (proved)
x+y=y -> x! + y! = y!.
end_of_list.

formulas(goals).
% Demonic Refinement Algebra (DRA22)
%BubbleSort
y*x + x*y = x*y  ->  (x+y)! + x! *y! = x! *y!.
end_of_list.
```

□

Teorema 6.2.689 (T_039_074)**.** *Na teoria Commutative dimonoid o resultado 6.1.40 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 258):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) + y = y -> x! * z + y = y.
(y*x + z) + y = y -> z * x! + y = y.
%isotonicity (proved)
x+y=y -> x! + y! = y!.
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
```

```
y + (x*y + z) = x*y + z -> y + x@ *z= x@ *z.
%isotonicity (proved)
x+y=y -> x@ +y@ = y@.
%isolation
x@ = x! + x@ *0.
end_of_list.

formulas(goals).
% Demonic Refinement Algebra (DRA33)
(x+y)@*0 =0 -> x@*0+y@*0=0.
end_of_list.
```

□

Teorema 6.2.690 (T_039_075). *Na teoria Commutative dimonoid o resultado 6.1.41 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 282):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) + y = y -> x! * z + y = y.
(y*x + z) + y = y -> z * x! + y = y.
%isotonicity (proved)
x+y=y -> x! + y! = y!.
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
y + (x*y + z) = x*y + z -> y + x@ *z= x@ *z.
%isotonicity (proved)
x+y=y -> x@ +y@ = y@.
%isolation
x@ = x! + x@ *0.
end_of_list.

formulas(goals).
% Demonic Refinement Algebra (DRA34)
y*x + x*(x+y)@= x*(x+y)@ -> ((x+y)@*0 =0 <- x@*0+y@*0=0).
end_of_list.
```

□

Teorema 6.2.691 (T_039_078). *Na teoria Commutative dimonoid o resultado 6.1.42 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 351):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
Commutative dimonoid
```

```
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
%inequality-axioms
x <= x.
x<=y & y<=z -> x<=z.
x<=y -> x+z<=y+z.
x<=y -> z+x<=z+y.
x<=y -> x*z<=y*z.
x<=y -> z*x<=z*y.
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) <= y -> x! * z <= y.
(y*x + z) <= y -> z * x! <= y.
%isotonicity (proved)
x<=y -> x! <= y!.
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
y <= x*y + z -> y <= x@ *z.
%isotonicity (proved)
x<=y -> x@ <= y@.
%isolation
x@ = x! + x@ *0.
%additional lemmas DRA17-19
x@*(y*x@)@ = (x+y)@.
x*(y*x)@ = (x*y)@*x.
end_of_list.

formulas(goals).
% Demonic Refinement Algebra (DRA37)
(x+y)@ <= (x!*y)@*x@.
(x!*y)@*x@ <= (x+y)@.
end_of_list.
```

□

Teorema 6.2.692 (T_039_276). *Na teoria Commutative dimonoid o resultado 6.1.45 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 325):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
```

```
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                       % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                 % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS10)
all x all p all q (test(p) & test(q) -> (fd(x,p) + q = q -> bb(x,q) + p = bb(x,q))).
end_of_list.
```

□

Teorema 6.2.693 (T_039_278). *Na teoria Commutative dimonoid o resultado 6.1.46 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 257):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                        % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                 % locality

% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                       % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                 % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
```

```
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS12)
all x all p all q (test(p) & test(q) -> p*fd(x,q)=fd(p*x,q)).
end_of_list.
```

□

Teorema 6.2.694 (T_039_282)**.** *Na teoria Commutative dimonoid o resultado 6.1.47 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 240):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                        % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).
```

```
%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS16)
all x all p all q (test(p) & test(q) -> (p*fd(x,q)=0 <-> q*bd(x,p)=0)).
end_of_list.
```

□

Teorema 6.2.695 (T_039_285). *Na teoria Commutative dimonoid o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 489):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.696 (T_039_287)**.** *Na teoria Commutative dimonoid o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 417):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                   % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

Teorema 6.2.697 (T_039_311)**.** *Na teoria Commutative dimonoid o resultado 6.1.65 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 246):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.

% proved lemata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dom(x+y)=dom(x)+dom(y).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS5)
all x all p all q (test(p) & test(q) -> fd(x,p)+fd(x,q) = fd(x,p+q)).
end_of_list.
```

□

Teorema 6.2.698 (T_039_362). *Na teoria Commutative dimonoid o resultado 6.1.75 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 366):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
```

```
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% proved lemmas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA27)
x;T=x ->
(x*y);z+(x*(y;z))=x*(y;z)
&
(x*y);z+(x*(y;z))=(x*y);z.
end_of_list.
```

□

Teorema 6.2.699 (T_039_385)**.** *Na teoria Commutative dimonoid o resultado 6.1.81 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 215):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
```

```
% Relation Algebra (RA48)
1<= syq(x,x).
end_of_list.
```

□

Teorema 6.2.700 (T_039_389). *Na teoria Commutative dimonoid o resultado 6.1.84 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 187):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA51)
x<=1 -> (x;T)' = (1*(x'));T.
end_of_list.
```

□

Teorema 6.2.701 (T_039_390). *Na teoria Commutative dimonoid o resultado 6.1.85 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 160):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
```

```
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA52)
x@;x<=1 & 1<=x;x@ -> x;syq(y,z) = syq(y;x@,z).
end_of_list.
```

□

Teorema 6.2.702 (T_039_391)**.** *Na teoria Commutative dimonoid o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 207):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
```

```
% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.703 (T_039_394)**.** *Na teoria Commutative dimonoid o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 212):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

Teorema 6.2.704 (T_039_395)**.** *Na teoria Commutative dimonoid o resultado 6.1.90 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 249):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Commutative dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA57)
(x;T);x = x -> (x;(x')@);x = 0.
end_of_list.
```

□

6.2.36 Complemented lattices

Teorema 6.2.705 (T_004_348)**.** *Na teoria Complemented lattices o resultado 6.1.69 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 176):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
```

```
% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA14)
(x;y)*(x;z)' = x;(y*z')*(x;z)' -> (x;y)'+(x;z) = (x;(y*z'))'+x;z.
end_of_list.
```

□

Teorema 6.2.706 (T_004_351). *Na teoria Complemented lattices o resultado 6.1.70 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 150):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
```

```
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA17)
x;T=x -> (x*1);y=x*y.
end_of_list.
```

□

Teorema 6.2.707 (T_004_355)**.** *Na teoria Complemented lattices o resultado 6.1.71 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 150):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
end_of_list.

formulas(goals).
% Relation Algebra (RA20)
x<=1 -> (x;T)*y=x;y.
end_of_list.
```

□

Teorema 6.2.708 (T_004_358)**.** *Na teoria Complemented lattices o resultado 6.1.73 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 153):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA23)
x + 1 = 1 & y + 1 = 1 ->
(x ; z * y ; z) + (x * y) ; z = (x * y) ; z
& (x ; z * y ; z) + (x * y) ; z = x ; z * y ; z.
end_of_list.
```

□

Teorema 6.2.709 (T_004_359)**.** *Na teoria Complemented lattices o resultado 6.1.74 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 157):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
```

```
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA24)
x+1=1 -> (x;y)*z' = (x;y)*(x;z)'.
end_of_list.
```

□

Teorema 6.2.710 (T_004_363)**.** *Na teoria Complemented lattices o resultado 6.1.76 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 156):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
```

```
end_of_list.

formulas(goals).
% Relation Algebra (RA28)
x;T=x ->   (y*x@);(x*z)+y;(x*z)=y;(x*z)
& (y*x@);(x*z)+(y*x@);z=(y*x@);z
& y;(x*z)+(y*x@);(x*z)=(y*x@);(x*z)
& (y*x@);z+(y*x@);(x*z)=(y*x@);(x*z).
end_of_list.
```

□

Teorema 6.2.711 (T_004_377). *Na teoria Complemented lattices o resultado 6.1.78 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 182):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

%greatest/smallest element
T = x+x'.
0 = x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
%row/column equivalence
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).

%Schroeder
x;y*z=0 <-> y*(x@;z)=0.
%Dedekind
(z;x*y) + (z*y;x@);(x*z@;y) = (z*y;x@);(x*z@;y) .
end_of_list.

formulas(goals).
% Relation Algebra (RA40)
syq(x',y') = syq(x,y).
syq(x,y)@ = syq(y,x).
```

```
end_of_list.
```

□

Teorema 6.2.712 (T_004_386)**.** *Na teoria Complemented lattices o resultado 6.1.82 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 155):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).
end_of_list.

formulas(goals).
% Relation Algebra (RA49)
syq(x,y) <= syq(z;x,z;y).
end_of_list.
```

□

Teorema 6.2.713 (T_004_391)**.** *Na teoria Complemented lattices o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 217):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.714 (T_004_394)**.** *Na teoria Complemented lattices o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 211):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
```

```
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

Teorema 6.2.715 (T_004_396)**.** *Na teoria Complemented lattices o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 180):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
```

```
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

Teorema 6.2.716 (T_004_400)**.** *Na teoria Complemented lattices o resultado 6.1.94 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 214):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.
```

```
formulas(goals).
% Relation Algebra (RA7)
(x;y)*z=0 -> y*(x@;z)=0.
end_of_list.
```

□

Teorema 6.2.717 (T_004_401)**.** *Na teoria Complemented lattices o resultado 6.1.95 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 197):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% bounded
0 + x = x.
1 + x = 1.

% complemented
all x exists y (x + y = 1 & x * y = 0).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA8)
(x;y)*z=0 <- y*(x@;z)=0.
end_of_list.
```

□

6.2.37 Complemented modular lattices

Teorema 6.2.718 (T_005_348)**.** *Na teoria Complemented modular lattices o resultado 6.1.69 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 167):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA14)
(x;y)*(x;z)' = x;(y*z')*(x;z)' -> (x;y)'+(x;z) = (x;(y*z'))'+x;z.
end_of_list.
```

□

Teorema 6.2.719 (T_005_351)**.** *Na teoria Complemented modular lattices o resultado 6.1.70 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 157):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
```

```
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA17)
x;T=x -> (x*1);y=x*y.
end_of_list.
```

□

Teorema 6.2.720 (T_005_355)**.** *Na teoria Complemented modular lattices o resultado 6.1.71 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 157):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.
```

```
% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
end_of_list.

formulas(goals).
% Relation Algebra (RA20)
x<=1 -> (x;T)*y=x;y.
end_of_list.
```

□

Teorema 6.2.721 (T_005_357). *Na teoria Complemented modular lattices o resultado 6.1.72 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 155):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
end_of_list.

formulas(goals).
% Relation Algebra (RA22)
x<=1 & y<=1 -> x;y=x*y.
end_of_list.
```

□

Teorema 6.2.722 (T_005_358)**.** *Na teoria Complemented modular lattices o resultado 6.1.73 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 162):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA23)
x + 1 = 1 & y + 1 = 1 ->
(x ; z * y ; z) + (x * y) ; z = (x * y) ; z
& (x ; z * y ; z) + (x * y) ; z = x ; z * y ; z.
end_of_list.
```

□

Teorema 6.2.723 (T_005_359)**.** *Na teoria Complemented modular lattices o resultado 6.1.74 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 160):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").
```

```
formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA24)
x+1=1 -> (x;y)*z' = (x;y)*(x;z)'.
end_of_list.
```

□

Teorema 6.2.724 (T_005_363)**.** *Na teoria Complemented modular lattices o resultado 6.1.76 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 165):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).
```

```
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA28)
x;T=x ->    (y*x@);(x*z)+y;(x*z)=y;(x*z)
& (y*x@);(x*z)+(y*x@);z=(y*x@);z
& y;(x*z)+(y*x@);(x*z)=(y*x@);(x*z)
& (y*x@);z+(y*x@);(x*z)=(y*x@);(x*z).
end_of_list.
```

□

Teorema 6.2.725 (T_005_386)**.** *Na teoria Complemented modular lattices o resultado 6.1.82 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 157):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
```

```
% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).
end_of_list.

formulas(goals).
% Relation Algebra (RA49)
syq(x,y) <= syq(z;x,z;y).
end_of_list.
```

□

Teorema 6.2.726 (T_005_391)**.** *Na teoria Complemented modular lattices o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 175):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
```

```
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.727 (T_005_396)**.** *Na teoria Complemented modular lattices o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 206):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Complemented modular lattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
all x exists y (x + y = 1 & x * y = 0).
% modular identity
((x * z) + y) * z = (x * z) + (y * z).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

6.2.38 Diassociative semigroup

Teorema 6.2.728 (T_040_285). *Na teoria Diassociative semigroup o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 284):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Diassociative semigroup
(x + y) + z = x + (y + z) & x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
(x * y) * z = (x + y) * z & (x + y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.729 (T_040_287). *Na teoria Diassociative semigroup o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 283):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
```

```
Diassociative semigroup
(x + y) + z = x + (y + z) & x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
(x * y) * z = (x + y) * z & (x + y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                   % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

6.2.39 Dimonoid

Teorema 6.2.730 (T_038_285)**.** *Na teoria Dimonoid o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 415):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.731 (T_038_287). *Na teoria Dimonoid o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 464):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
```

```
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

Teorema 6.2.732 (T_038_379). *Na teoria Dimonoid o resultado 6.1.79 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 173):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Dimonoid
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.


% Schroeder %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;y+z'=z' <-> y'+x@;z=y'.
x@;z+y'=y' <-> y;z@+x'=x'.

% order props %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x+y=y <-> x@+y@=y@.
x+y=y <-> x'+y'=x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA42)
(x;y);x+x=x <-> y+((x;x'@);x)'@=((x;x'@);x)'@.
```

```
end_of_list.
```

□

6.2.40 Disemigroup

Teorema 6.2.733 (T_035_074)**.** *Na teoria Disemigroup o resultado 6.1.40 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 249):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
Disemigroup
% Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
% Right disemigroup
% (x + y) + z = x + (y + z).
% (x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) + y = y -> x! * z + y = y.
(y*x + z) + y = y -> z * x! + y = y.
%isotonicity (proved)
x+y=y -> x! + y! = y!.
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
y + (x*y + z) = x*y + z -> y + x@ *z= x@ *z.
%isotonicity (proved)
x+y=y -> x@ +y@ = y@.
%isolation
x@ = x! + x@ *0.
end_of_list.

formulas(goals).
% Demonic Refinement Algebra (DRA33)
(x+y)@*0 =0 -> x@*0+y@*0=0.
end_of_list.
```

□

Teorema 6.2.734 (T_035_075)**.** *Na teoria Disemigroup o resultado 6.1.41 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 210):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
Disemigroup
% Left disemigroup
```

```
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
% Right disemigroup
% (x + y) + z = x + (y + z).
% (x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) + y = y -> x! * z + y = y.
(y*x + z) + y = y -> z * x! + y = y.
%isotonicity (proved)
x+y=y -> x! + y! = y!.
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
y + (x*y + z) = x*y + z -> y + x@ *z= x@ *z.
%isotonicity (proved)
x+y=y -> x@ +y@ = y@.
%isolation
x@ = x! + x@ *0.
end_of_list.

formulas(goals).
% Demonic Refinement Algebra (DRA34)
y*x + x*(x+y)@= x*(x+y)@ -> ((x+y)@*0 =0 <- x@*0+y@*0=0).
end_of_list.
```

□

Teorema 6.2.735 (T_035_078)**.** *Na teoria Disemigroup o resultado 6.1.42 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 471):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").
op(480, postfix, "@"). %omega

formulas(assumptions).
Disemigroup
% Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
% Right disemigroup
% (x + y) + z = x + (y + z).
% (x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
%inequality-axioms
x <= x.
x<=y & y<=z -> x<=z.
x<=y -> x+z<=y+z.
x<=y -> z+x<=z+y.
x<=y -> x*z<=y*z.
x<=y -> z*x<=z*y.
% standard axioms for finite iteration (star) %%%%%%%%%
1 + x * x! = x!.
1 + x! * x = x!.
(x*y + z) <= y -> x! * z <= y.
(y*x + z) <= y -> z * x! <= y.
%isotonicity (proved)
x<=y -> x! <= y!.
```

```
% omega axioms
x@ = x*x@ +1.
x@ = x@ *x+1.
y <= x*y + z -> y <= x@ *z.
%isotonicity (proved)
x<=y -> x@ <= y@.
%isolation
x@ = x! + x@ *0.
%additional lemmas DRA17-19
x@*(y*x@)@ = (x+y)@.
x*(y*x)@ = (x*y)@*x.
end_of_list.

formulas(goals).
% Demonic Refinement Algebra (DRA37)
(x+y)@ <= (x!*y)@*x@.
(x!*y)@*x@ <= (x+y)@.
end_of_list.
```

□

Teorema 6.2.736 (T_035_285). *Na teoria Disemigroup o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 423):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Disemigroup
% Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
% Right disemigroup
% (x + y) + z = x + (y + z).
% (x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
```

```
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.737 (T_035_287). *Na teoria Disemigroup o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 389):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Disemigroup
% Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
% Right disemigroup
% (x + y) + z = x + (y + z).
% (x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
```

```
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

Teorema 6.2.738 (T_035_362)**.** *Na teoria Disemigroup o resultado 6.1.75 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 202):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Disemigroup
% Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
% Right disemigroup
% (x + y) + z = x + (y + z).
% (x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% proved lemmas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA27)
x;T=x ->
(x*y);z+(x*(y;z))=x*(y;z)
&
(x*y);z+(x*(y;z))=(x*y);z.
end_of_list.
```

□

6.2.41 Doppelsemigroup

Teorema 6.2.739 (T_036_285)**.** *Na teoria Doppelsemigroup o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 245):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Doppelsemigroup
(x + y) * z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.740 (T_036_287)**.** *Na teoria Doppelsemigroup o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 235):

```
op(500, infix, "+").
op(490, infix, "*").
```

```
formulas(assumptions).
Doppelsemigroup
(x + y) * z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

6.2.42 Duplicial semigroup

Teorema 6.2.741 (T_041_285)**.** *Na teoria Duplicial semigroup o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 229):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Duplicial semigroup
(x + y) + z = x + (y + z).
(x * y) + z = x * (y + z).
(x * y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.
```

```
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.742 (T_041_287)**.** *Na teoria Duplicial semigroup o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 203):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Duplicial semigroup
(x + y) + z = x + (y + z).
(x * y) + z = x * (y + z).
(x * y) * z = x * (y * z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
```

```
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                            % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

6.2.43 Involutive lattices

Teorema 6.2.743 (T_010_391). *Na teoria Involutive lattices o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 176):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Involutive lattices
% lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% De Morgan involution
-(x * y) = -x + -y.
--x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.
```

```
% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.744 (T_010_394)**.** *Na teoria Involutive lattices o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 168):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Involutive lattices
% lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% De Morgan involution
-(x * y) = -x + -y.
--x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
```

```
end_of_list.
```

□

Teorema 6.2.745 (T_010_396)**.** *Na teoria Involutive lattices o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 196):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Involutive lattices
% lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% De Morgan involution
-(x * y) = -x + -y.
--x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

6.2.44 Left disemigroup

Teorema 6.2.746 (T_033_285)**.** *Na teoria Left disemigroup o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 229):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                   % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.747 (T_033_287). *Na teoria Left disemigroup o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 188):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
%subsumption order
x <= y <-> x+y=y.
```

```
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                        % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                        % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

Teorema 6.2.748 (T_033_394). *Na teoria Left disemigroup o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 224):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.
```

```
% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

6.2.45 Multiplicative semilattices

Teorema 6.2.749 (T_011_283). *Na teoria Multiplicative semilattices o resultado 6.1.48 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 154):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Multiplicative semilattices
% semilattice
(x + y) + z = x + (y + z).
x + y = y + x.
x + x = x.

%
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.
end_of_list.

formulas(goals).
```

```
% Modal I-Semiring (MIS17)
all p all q(test(p) & test(q) -> fd(p,q) = p*q).
end_of_list.
```

□

Teorema 6.2.750 (T_011_311). *Na teoria Multiplicative semilattices o resultado 6.1.65 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 237):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Multiplicative semilattices
% semilattice
(x + y) + z = x + (y + z).
x + y = y + x.
x + x = x.

%
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.

% proved lemata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dom(x+y)=dom(x)+dom(y).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS5)
all x all p all q (test(p) & test(q) -> fd(x,p)+fd(x,q) = fd(x,p+q)).
end_of_list.
```

□

6.2.46 Near-ring

Teorema 6.2.751 (T_013_291). *Na teoria Near-ring o resultado 6.1.52 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 170):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS24)
fd(x,d(y))*d(z) = 0 <- d(y)*bd(x,d(z)) = 0.
end_of_list.
```

□

6.2.47 Near-semiring

6.2.48 Neardistributive lattices

Teorema 6.2.752 (T_015_391)**.** *Na teoria Neardistributive lattices o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 183):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Neardistributive lattices
% lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
```

```
(x + y) * x = x.
(x * y) + x = x.

%
x * (y + z) = x * (y + (x * (z + (x * y)))).
x + (y * z) = x + (y * (x + (z * (x + y)))).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.753 (T_015_394). *Na teoria Neardistributive lattices o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 220):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Neardistributive lattices
% lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

%
x * (y + z) = x * (y + (x * (z + (x * y)))).
x + (y * z) = x + (y * (x + (z * (x + y)))).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
```

```
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

Teorema 6.2.754 (T_015_396)**.** *Na teoria Neardistributive lattices o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 201):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Neardistributive lattices
% lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

%
x * (y + z) = x * (y + (x * (z + (x * y)))).
x + (y * z) = x + (y * (x + (z * (x + y)))).

% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.
```

```
formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

6.2.49 Ockham algebras

Teorema 6.2.755 (T_016_276). *Na teoria Ockham algebras o resultado 6.1.45 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 326):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                        % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
```

```
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS10)
all x all p all q (test(p) & test(q) -> (fd(x,p) + q = q -> bb(x,q) + p = bb(x,q))).
end_of_list.
```

□

Teorema 6.2.756 (T_016_284). *Na teoria Ockham algebras o resultado 6.1.49 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 155):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS18)
all p all q(test(p) & test(q) -> fd(p,fb(p,q)) = fd(p,q)).
end_of_list.
```

□

Teorema 6.2.757 (T_016_292). *Na teoria Ockham algebras o resultado 6.1.53 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 185):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS25)
fb(x,d(y)) + d(z) = 1 <- d(y) + bb(x,d(z)) = 1.
end_of_list.
```

□

Teorema 6.2.758 (T_016_293). *Na teoria Ockham algebras o resultado 6.1.54 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 203):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS26)
fb(x,d(y)) + d(z) = 1 -> d(y) + bb(x,d(z)) = 1.
end_of_list.
```

□

Teorema 6.2.759 (T_016_294)**.** *Na teoria Ockham algebras o resultado 6.1.55 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 187):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
```

```
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS27)
fd(x,d(y))+d(z) = d(z) <- d(y)+bb(x,d(z)) = bb(x,d(z)).
end_of_list.
```

□

Teorema 6.2.760 (T_016_295). *Na teoria Ockham algebras o resultado 6.1.56 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 197):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
```

```
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS28)
fd(x,d(y))+d(z) = d(z) -> d(y)+bb(x,d(z)) = bb(x,d(z)).
end_of_list.
```

□

Teorema 6.2.761 (T_016_296)**.** *Na teoria Ockham algebras o resultado 6.1.57 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 191):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).
```

```
%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS29)
bd(x,d(y))+d(z) = d(z) <- d(y)+fb(x,d(z)) = fb(x,d(z)).
end_of_list.
```

□

Teorema 6.2.762 (T_016_298)**.** *Na teoria Ockham algebras o resultado 6.1.58 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 182):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
```

```
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS30)
bd(x,d(y))+d(z) = d(z) -> d(y)+fb(x,d(z)) = fb(x,d(z)).
end_of_list.
```

□

Teorema 6.2.763 (T_016_299). *Na teoria Ockham algebras o resultado 6.1.59 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 180):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
```

```
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS31)
fd(x,bb(x,d(y)))+d(y)=d(y).
end_of_list.
```

□

Teorema 6.2.764 (T_016_300). *Na teoria Ockham algebras o resultado 6.1.60 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 180):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS32)
d(y)+bb(x,fd(x,d(y))) = bb(x,fd(x,d(y))).
end_of_list.
```

□

Teorema 6.2.765 (T_016_301). *Na teoria Ockham algebras o resultado 6.1.61 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 179):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS33)
bd(x,fb(x,d(y)))+d(y)=d(y).
end_of_list.
```

□

Teorema 6.2.766 (T_016_302). *Na teoria Ockham algebras o resultado 6.1.62 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 180):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS34)
d(y)+fb(x,bd(x,d(y))) = fb(x,bd(x,d(y))).
end_of_list.
```

□

Teorema 6.2.767 (T_016_305)**.** *Na teoria Ockham algebras o resultado 6.1.63 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 150):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
```

```
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS37)
fd(x,d(y)+d(z)) = fd(x,d(y))+fd(x,d(z)).
end_of_list.
```

□

Teorema 6.2.768 (T_016_306). *Na teoria Ockham algebras o resultado 6.1.64 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 156):

```
op(500, infix, "+").
op(490, infix, "*").
op(500, infix, "/").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
```

```
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
%boolean domain axioms
a(x)*x = 0.
a(x*y)+a(x*a(a(y))) = a(x*a(a(y))).
a(a(x))+a(x) = 1.
d(x) = a(a(x)).

%boolean codomain axioms
x*ca(x) = 0.
ca(x*y)+ca(ca(ca(x))*y) = ca(ca(ca(x))*y).
ca(ca(x))+ca(x) = 1.
cd(x) = ca(ca(x)).

%diamond and box axioms
cp(x) = a(d(x)).
%domain difference
x/y = d(x)*a(y).
%forward diamond
fd(x,y) = d(x*d(y)).
%backward diamond
bd(x,y) = cd(cd(y)*x).
%forward box
fb(x,y) = cp(fd(x,cp(y))).
%backward box
bb(x,y) = cp(bd(x,cp(y))).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS38)
fb(x,d(y)*d(z)) = fb(x,d(y))*fb(x,d(z)).
end_of_list.
```

□

Teorema 6.2.769 (T_016_311). *Na teoria Ockham algebras o resultado 6.1.65 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 230):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).
```

```
%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
%  all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
%  -test(y) -> fb(x,y) = 0.

% proved lemata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dom(x+y)=dom(x)+dom(y).
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS5)
all x all p all q (test(p) & test(q) -> fd(x,p)+fd(x,q) = fd(x,p+q)).
end_of_list.
```

□

Teorema 6.2.770 (T_016_312)**.** *Na teoria Ockham algebras o resultado 6.1.66 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 182):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
```

```
% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS6)
all x all y all p (test(p) -> fb(x,fb(y,p)) = fb(x*y,p)).
end_of_list.
```

□

Teorema 6.2.771 (T_016_314)**.** *Na teoria Ockham algebras o resultado 6.1.67 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 210):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS8)
all x all p all q (test(p) & test(q) -> fb(x,p)*fb(x,q) = fb(x,p*q)).
```

```
end_of_list.
```

□

Teorema 6.2.772 (T_016_315). *Na teoria Ockham algebras o resultado 6.1.68 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 296):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Ockham algebras
% bounded distributive lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
0 + x = x.
1 + x = 1.
x * (y + z) = (x * y) + (x * z).

% dual endomorphism
(x * y)' = x' + y'.
(x + y)' = x' * y'.
0' = 1.
1' = 0.
% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality

% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                        % codomain is test
all p (test(p) -> cod(x*p) = cod(x)*p).
x*cod(x) = x.
cod(x*y) = cod(cod(x)*y).                  % locality

% standard axioms for forward box and diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).

%additional axioms, since fd/fb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.

% standard axiom for backward box and diamond %%%%%%%%%
all p (test(p) -> bd(x,p)=cod(p*x)).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since bd/bb have to be total functions
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
```

```
% Modal I-Semiring (MIS9)
all x all p all q (test(p) & test(q) -> (fd(x,p) + q = q <- bb(x,q) + p = bb(x,q))).
end_of_list.
```

□

6.2.50 Ortholattices

Teorema 6.2.773 (T_017_386)**.** *Na teoria Ortholattices o resultado 6.1.82 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 169):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Ortholattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
x + x' = 1.
x * x' = 0.
% de morgan
(x + y)' = x' * y'.
(x * y)' = x' + y'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).
end_of_list.

formulas(goals).
% Relation Algebra (RA49)
syq(x,y) <= syq(z;x,z;y).
end_of_list.
```

□

Teorema 6.2.774 (T_017_391). *Na teoria Ortholattices o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 173):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Ortholattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
x + x' = 1.
x * x' = 0.
% de morgan
(x + y)' = x' * y'.
(x * y)' = x' + y'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.775 (T_017_396). *Na teoria Ortholattices o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 222):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
```

```
op(300, postfix, "@").

formulas(assumptions).
Ortholattices
% lattice
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + y = y + x.
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.
% bounded
0 + x = x.
1 + x = 1.
% complemented
x + x' = 1.
x * x' = 0.
% de morgan
(x + y)' = x' * y'.
(x * y)' = x' + y'.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

6.2.51 Quandles

Teorema 6.2.776 (T_020_374)**.** *Na teoria Quandles o resultado 6.1.77 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 737):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
```

```
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

%bounded elements
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA38)
(x;T);x =x <-> (x;x'@);x=0.
end_of_list.
```

□

Teorema 6.2.777 (T_020_377)**.** *Na teoria Quandles o resultado 6.1.78 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 168):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

%greatest/smallest element
T = x+x'.
0 = x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
%row/column equivalence
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).
```

```
%Schroeder
x;y*z=0 <-> y*(x@;z)=0.
%Dedekind
(z;x*y) + (z*y;x@);(x*z@;y) = (z*y;x@);(x*z@;y) .
end_of_list.

formulas(goals).
% Relation Algebra (RA40)
syq(x',y') = syq(x,y).
syq(x,y)@ = syq(y,x).
end_of_list.
```

□

Teorema 6.2.778 (T_020_379). *Na teoria Quandles o resultado 6.1.79 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 153):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.


% Schroeder %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;y+z'=z' <-> y'+x@;z=y'.
x@;z+y'=y' <-> y;z@+x'=x'.

% order props %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x+y=y <-> x@+y@=y@.
x+y=y <-> x'+y'=x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA42)
(x;y);x+x=x <-> y+((x;x'@);x)'@=((x;x'@);x)'@.
end_of_list.
```

□

Teorema 6.2.779 (T_020_384). *Na teoria Quandles o resultado 6.1.80 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 163):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA47)
x;(y*z) = x;y * x;z -> x@;x<=1.
end_of_list.
```

□

Teorema 6.2.780 (T_020_386)**.** *Na teoria Quandles o resultado 6.1.82 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 269):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

syq(x,y) = (x@;y')'*(x'@;y)'.
% row/column equivalence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
roweq(x) = syq(x@,x@).
coleq(x) = syq(x,x).
end_of_list.

formulas(goals).
% Relation Algebra (RA49)
syq(x,y) <= syq(z;x,z;y).
end_of_list.
```

□

Teorema 6.2.781 (T_020_388)**.** *Na teoria Quandles o resultado 6.1.83 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 164):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
```

```
%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA50)
x<=(x;x@);x.
end_of_list.
```

□

Teorema 6.2.782 (T_020_389). *Na teoria Quandles o resultado 6.1.84 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 213):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA51)
x<=1 -> (x;T)' = (1*(x'));T.
end_of_list.
```

□

Teorema 6.2.783 (T_020_391). *Na teoria Quandles o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 231):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.784 (T_020_392)**.** *Na teoria Quandles o resultado 6.1.87 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 315):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
```

```
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA54)
x!=0 <-> x@;x!=0.
end_of_list.
```

□

Teorema 6.2.785 (T_020_393)**.** *Na teoria Quandles o resultado 6.1.88 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 423):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.
```

```
formulas(goals).
% Relation Algebra (RA55)
(x;T)' = (x;T)';T.
end_of_list.
```

□

Teorema 6.2.786 (T_020_396)**.** *Na teoria Quandles o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 268):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

Teorema 6.2.787 (T_020_397)**.** *Na teoria Quandles o resultado 6.1.92 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 227):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").
```

```
formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA59)
x+y<=z <-> (x<=z & y<=z).
end_of_list.
```

□

Teorema 6.2.788 (T_020_399)**.** *Na teoria Quandles o resultado 6.1.93 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 428):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
```

```
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA60)
x = (1*x;x@);x.
end_of_list.
```

□

Teorema 6.2.789 (T_020_400)**.** *Na teoria Quandles o resultado 6.1.94 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 589):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA7)
(x;y)*z=0 -> y*(x@;z)=0.
end_of_list.
```

□

Teorema 6.2.790 (T_020_401)**.** *Na teoria Quandles o resultado 6.1.95 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 545):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.
end_of_list.

formulas(goals).
% Relation Algebra (RA8)
(x;y)*z=0 <- y*(x@;z)=0.
end_of_list.
```

□

Teorema 6.2.791 (T_020_402)**.** *Na teoria Quandles o resultado 6.1.96 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 152):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Quandles
% left-selfdistributive
x + (y + z) = (x + y) + (x + z).
% right-selfdistributive
(x * y) * z = (x * z) * (y * z).
%
(x + y) * x = y.
x + (y * x) = y.
x + x = x.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
```

```
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
end_of_list.

formulas(goals).
% Relation Algebra (RA9)
(y;x)';x@ <= y'.
end_of_list.
```

□

6.2.52 Right disemigroup

Teorema 6.2.792 (T_034_285)**.** *Na teoria Right disemigroup o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 311):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Right disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
```

```
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.793 (T_034_287). *Na teoria Right disemigroup o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 266):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Right disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

Teorema 6.2.794 (T_034_362)**.** *Na teoria Right disemigroup o resultado 6.1.75 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 435):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Right disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% proved lemmas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA27)
x;T=x ->
(x*y);z+(x*(y;z))=x*(y;z)
&
(x*y);z+(x*(y;z))=(x*y);z.
end_of_list.
```

□

6.2.53 Shells

Teorema 6.2.795 (T_025_362)**.** *Na teoria Shells o resultado 6.1.75 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 216):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Shells
0 + x = x.
x + 0 = x.
```

```
1 * x = x.
x * 1 = x.
0 * x = 0.
x * 0 = 0.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% proved lemmas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA27)
x;T=x ->
(x*y);z+(x*(y;z))=x*(y;z)
&
(x*y);z+(x*(y;z))=(x*y);z.
end_of_list.
```

□

Teorema 6.2.796 (T_025_391)**.** *Na teoria Shells o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 308):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Shells
0 + x = x.
x + 0 = x.
1 * x = x.
x * 1 = x.
0 * x = 0.
x * 0 = 0.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
```

```
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.797 (T_025_394). *Na teoria Shells o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 250):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Shells
0 + x = x.
x + 0 = x.
1 * x = x.
x * 1 = x.
0 * x = 0.
x * 0 = 0.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

Teorema 6.2.798 (T_025_396). *Na teoria Shells o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 281):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Shells
0 + x = x.
x + 0 = x.
1 * x = x.
x * 1 = x.
0 * x = 0.
x * 0 = 0.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

6.2.54 Skew lattices

Teorema 6.2.799 (T_026_362)**.** *Na teoria Skew lattices o resultado 6.1.75 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 194):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Skew lattices
% band
(x + y) + z = x + (y + z).
x + x = x.

% band
(x * y) * z = x * (y * z).
x * x = x.

% absorption
```

```
x * (x + y) = x & x = x + (x * y).
(x + y) * y = y & y = (x * y) + y.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

% proved lemmas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA27)
x;T=x ->
(x*y);z+(x*(y;z))=x*(y;z)
&
(x*y);z+(x*(y;z))=(x*y);z.
end_of_list.
```

□

Teorema 6.2.800 (T_026_391)**.** *Na teoria Skew lattices o resultado 6.1.86 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 207):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Skew lattices
% band
(x + y) + z = x + (y + z).
x + x = x.

% band
(x * y) * z = x * (y * z).
x * x = x.

% absorption
x * (x + y) = x & x = x + (x * y).
(x + y) * y = y & y = (x * y) + y.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.
```

```
% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA53)
x;(y';z)' <= ((x;y)';z)'.
end_of_list.
```

□

Teorema 6.2.801 (T_026_394)**.** *Na teoria Skew lattices o resultado 6.1.89 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 171):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Skew lattices
% band
(x + y) + z = x + (y + z).
x + x = x.

% band
(x * y) * z = x * (y * z).
x * x = x.

% absorption
x * (x + y) = x & x = x + (x * y).
(x + y) * y = y & y = (x * y) + y.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

demonic_composition(x,y) = (x;y)*(x;(y;T)')'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA56)
demonic_composition(x,demonic_composition(y,z)) = demonic_composition(demonic_composition(x,y),z).
end_of_list.
```

□

Teorema 6.2.802 (T_026_396)**.** *Na teoria Skew lattices o resultado 6.1.91 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 245):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Skew lattices
% band
(x + y) + z = x + (y + z).
x + x = x.

% band
(x * y) * z = x * (y * z).
x * x = x.

% absorption
x * (x + y) = x & x = x + (x * y).
(x + y) * y = y & y = (x * y) + y.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA58)
x<=y*z <-> (x<=y & x<=z).
end_of_list.
```

□

Teorema 6.2.803 (T_026_397)**.** *Na teoria Skew lattices o resultado 6.1.92 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 223):

```
op(500, infix,   "+").
op(500, infix,   "*").
op(300, postfix, "'").
op(450, infix,   ";").
op(300, postfix, "@").

formulas(assumptions).
Skew lattices
% band
```

```
(x + y) + z = x + (y + z).
x + x = x.

% band
(x * y) * z = x * (y * z).
x * x = x.

% absorption
x * (x + y) = x & x = x + (x * y).
(x + y) * y = y & y = (x * y) + y.
% composition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x;(y;z) = (x;y);z.
x;1=x.
(x+y);z = (x;z)+(y;z).

% converse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(x@)@ = x.
(x+y)@ = x@ + y@.
(x;y)@ = y@ ; x@.
x@;(x;y)' +y' =y'.

% ordering %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x<=y <-> x+y=y.

% identities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T=x+x'.
0=x*x'.

%Dedekind
(x;y*z) + (x*(z;y@);(y*x@;z)) = x*(z;y@);(y*x@;z).
end_of_list.

formulas(goals).
% Relation Algebra (RA59)
x+y<=z <-> (x<=z & y<=z).
end_of_list.
```

□

6.2.55 Strong doppelsemigroup

Teorema 6.2.804 (T_037_259)**.** *Na teoria Strong doppelsemigroup o resultado 6.1.43 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 344):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").

formulas(assumptions).
Strong doppelsemigroup
(x + y) * z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).

x + (y * z) = x * (y + z).
%subsumption order
x <= y <-> x+y=y.
%right superdistributivity
x*y + x*z <= x*(y + z).
% lazy Kleene algebras %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%finite iteration (star)
%left unfold
1 + x * x! <= x!.
%left induction
```

```
x*y + z <= y -> x! * z <= y.
end_of_list.

formulas(goals).
% Left Kleene Algebra (LKA1)
x <= x!.
end_of_list.
```

□

Teorema 6.2.805 (T_037_272). *Na teoria Strong doppelsemigroup o resultado 6.1.44 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 363):

```
op(500, infix, "+").
op(490, infix, "*").
op(480, postfix, "!").

formulas(assumptions).
Strong doppelsemigroup
(x + y) * z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).

x + (y * z) = x * (y + z).
%subsumption order
x <= y <-> x+y=y.
%right superdistributivity
x*y + x*z <= x*(y + z).
% lazy Kleene algebras %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%finite iteration (star)
%left unfold
1 + x * x! <= x!.
%left induction
x*y + z <= y -> x! * z <= y.
end_of_list.

formulas(goals).
% Left Kleene Algebra (LKA7)
x!*x <= x*x!.
end_of_list.
```

□

Teorema 6.2.806 (T_037_285). *Na teoria Strong doppelsemigroup o resultado 6.1.50 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 390):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Strong doppelsemigroup
(x + y) * z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).

x + (y * z) = x * (y + z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
```

```
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
-test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS19)
all p (test(p) -> bd(x,fb(x,p)) <= p).
end_of_list.
```

□

Teorema 6.2.807 (T_037_287)**.** *Na teoria Strong doppelsemigroup o resultado 6.1.51 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9 (Max length proof: 210):

```
op(500, infix, "+").
op(490, infix, "*").

formulas(assumptions).
Strong doppelsemigroup
(x + y) * z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).

x + (y * z) = x * (y + z).
%subsumption order
x <= y <-> x+y=y.

% standard axioms for tests %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p)*c(q) & c(p*q) = c(p)+c(q)).
all p (test(p) -> p*c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).

%additional axioms, since c() has to be a total function
-test(x) -> c(x) = 0.

% standard axioms for domain %%%%%%%%%%%%%%%%%%%%%%%%%%%
test(dom(x)).                          % domain is test
```

```
all p (test(p) -> dom(p*x) = p*dom(x)).
dom(x)*x = x.
dom(x*y) = dom(x*dom(y)).                  % locality
% standard axioms for codomain %%%%%%%%%%%%%%%%%%%%%%%%%%
test(cod(x)).                          % codomain is test
all p (test(p) -> cod(x*p) <= p).
x <= x*cod(x).
cod(cod(x)*y) <= cod(x*y).
cod(x) + cod(y) <= cod(x+y).
% standard axioms for forward and backward diamond %%%%%%%%%
all p (test(p) -> fd(x,p)=dom(x*p)).
all p (test(p) -> bd(x,p)=cod(p*x)).
% standard axioms for forward and backward box %%%%%%%%%
all p (test(p) -> fb(x,p)=c(fd(x,c(p)))).
%  all p (test(p) -> bb(x,p)=c(bd(x,c(p)))).

%additional axioms, since fd/fb nad bd/bb have to be total functions
-test(y) -> fd(x,y) = 0.
-test(y) -> fb(x,y) = 0.
-test(y) -> bd(x,y) = 0.
%  -test(y) -> bb(x,y) = 0.
end_of_list.

formulas(goals).
% Modal I-Semiring (MIS20)
all p (test(p) -> p <= fb(x,bd(x,p))).
end_of_list.
```

□

6.2.56 Commutative regular rings

Teorema 6.2.808. *Na teoria Commutative regular rings o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
```

```
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.809. *Na teoria Commutative regular rings o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.810. *Na teoria Commutative regular rings o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
```

```
end_of_list.
```

□

Lema 6.2.183. *Na teoria Commutative regular rings o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.184. *Na teoria Commutative regular rings o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
```

```
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.185. *Na teoria Commutative regular rings o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% MNMN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.2.186. *Na teoria Commutative regular rings o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% NPJSSO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.2.187. *Na teoria Commutative regular rings o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% NSNSM
```

```
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = (x * y)' * (y cap x).
end_of_list.
```

□

Lema 6.2.188. *Na teoria Commutative regular rings o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.189. *Na teoria Commutative regular rings o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.190. *Na teoria Commutative regular rings o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
```

```
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.57 Division ring

Teorema 6.2.811. *Na teoria Division ring o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.2.191. *Na teoria Division ring o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
```

```
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.192. *Na teoria Division ring o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
```

```
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.193. *Na teoria Division ring o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% MNMN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.2.194. *Na teoria Division ring o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
```

```
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NPJSSO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.2.195. *Na teoria Division ring o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NSNSM
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = (x * y)' * (y cap x).
```

```
end_of_list.
```

□

Lema 6.2.196. *Na teoria Division ring o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.197. *Na teoria Division ring o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
```

```
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.198. *Na teoria Division ring o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Division ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.58 Fields

Teorema 6.2.812. *Na teoria Fields o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.813. *Na teoria Fields o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.
```

```
x != 0 -> exists y (x * y = 1).
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.814. *Na teoria Fields o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.2.199. *Na teoria Fields o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
```

```
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.200. *Na teoria Fields o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
```

```
end_of_list.
```

□

Lema 6.2.201. *Na teoria Fields o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% MNMN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.2.202. *Na teoria Fields o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
```

```
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NPJSSO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.2.203. *Na teoria Fields o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NSNSM
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
```

```
x' = (x * y)' * (y cap x).
end_of_list.
```

□

Lema 6.2.204. *Na teoria Fields o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.205. *Na teoria Fields o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
```

```
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.206. *Na teoria Fields o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
```

```
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.59 Integral domains

Teorema 6.2.815. *Na teoria Integral domains o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.816. *Na teoria Integral domains o resultado 6.1.97 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.
```

```
(x * y = 0) -> (x = 0 | y = 0).
% PLM2
%% power-like functions
f(x) = x * t(x).
t(x) * x = x * t(x).
t(x * y) * x = x * t(y * x).
t(x) * t(x) = t(x * x).
t(x)' = t(x').
f(x) * ((y * y) * t(x)) = y * ((f(x) * t(x)) * y).
end_of_list.
formulas(goals).
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.817. *Na teoria Integral domains o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.2.207. *Na teoria Integral domains o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
```

```
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.208. *Na teoria Integral domains o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.209. *Na teoria Integral domains o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% MNMN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.2.210. *Na teoria Integral domains o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% NPJSSO
```

```
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.2.211. *Na teoria Integral domains o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% NSNSM
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = (x * y)' * (y cap x).
end_of_list.
```

□

Lema 6.2.212. *Na teoria Integral domains o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.213. *Na teoria Integral domains o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
```

```
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.214. *Na teoria Integral domains o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Integral domains
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

(x * y = 0) -> (x = 0 | y = 0).
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.60 Near-fields

Teorema 6.2.818. *Na teoria Near-fields o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.2.215. *Na teoria Near-fields o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

6.2.61 Near-ring

Teorema 6.2.819. *Na teoria Near-ring o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.2.216. *Na teoria Near-ring o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.217. *Na teoria Near-fields o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").
```

```
formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.218. *Na teoria Near-fields o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% MNMN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.2.219. *Na teoria Near-fields o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% NPJSSO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.2.220. *Na teoria Near-fields o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% NSNSM
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
```

```
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = (x * y)' * (y cap x).
end_of_list.
```

□

Lema 6.2.221. *Na teoria Near-fields o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.222. *Na teoria Near-fields o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
```

```
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.223. *Na teoria Near-fields o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-fields
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

Lema 6.2.224. *Na teoria Near-ring o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.225. *Na teoria Near-ring o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% MNMN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.2.226. *Na teoria Near-ring o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% NPJSSO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.2.227. *Na teoria Near-ring o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% NSNSM
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
```

```
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = (x * y)' * (y cap x).
end_of_list.
```

□

Lema 6.2.228. *Na teoria Near-ring o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.229. *Na teoria Near-ring o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
```

```
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.230. *Na teoria Near-ring o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-ring
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.62 Near-rings with identity

Teorema 6.2.820. *Na teoria Near-rings with identity o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
```

```
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.2.231. *Na teoria Near-rings with identity o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.232. *Na teoria Near-rings with identity o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
```

```
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.233. *Na teoria Near-rings with identity o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% MNMN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.2.234. *Na teoria Near-rings with identity o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% NPJSSO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.2.235. *Na teoria Near-rings with identity o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% NSNSM
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
```

```
x' = (x * y)' * (y cap x).
end_of_list.
```

□

Lema 6.2.236. *Na teoria Near-rings with identity o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.237. *Na teoria Near-rings with identity o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
```

```
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.238. *Na teoria Near-rings with identity o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Near-rings with identity
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.63 Skew-fields

Teorema 6.2.821. *Na teoria Skew-fields o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
```

```
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.2.239. *Na teoria Skew-fields o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.240. *Na teoria Skew-fields o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.241. *Na teoria Skew-fields o resultado 6.1.14 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.
```

```
x != 0 -> exists y (x * y = 1).
% MNMN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = (y cap x)'.
end_of_list.
```

□

Lema 6.2.242. *Na teoria Skew-fields o resultado 6.1.15 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NPJSSO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' + ((y cup x) * (y * x)) = 1.
end_of_list.
```

□

Lema 6.2.243. *Na teoria Skew-fields o resultado 6.1.16 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NSNSM
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = (x * y)' * (y cap x).
end_of_list.
```

□

Lema 6.2.244. *Na teoria Skew-fields o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% NNSSNN
x cup y = x + (y * x).
```

```
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.245. *Na teoria Skew-fields o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.246. *Na teoria Skew-fields o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Skew-fields
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

0 != 1.

x != 0 -> exists y (x * y = 1).
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.64 Domain semigroup

Lema 6.2.247. *Na teoria Domain semigroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain semigroup
(x * y) * z = x * (y * z).
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
```

```
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.248. *Na teoria Domain semigroup o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain semigroup
(x * y) * z = x * (y * z).
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.249. *Na teoria Domain semigroup o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain semigroup
(x * y) * z = x * (y * z).
x = d(x) * x.
d(x * y) = d(x * d(y)).
```

```
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.250. *Na teoria Domain semigroup o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain semigroup
(x * y) * z = x * (y * z).
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.251. *Na teoria Domain semigroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain semigroup
(x * y) * z = x * (y * z).
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.65 Domain monoid

Lema 6.2.252. *Na teoria Domain monoid o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
```

```
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
end_of_list.
```

□

Lema 6.2.253. *Na teoria Domain monoid o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.254. *Na teoria Domain monoid o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
```

```
d(x + y) = d(x) + d(y).
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.255. *Na teoria Domain monoid o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.256. *Na teoria Domain monoid o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Domain monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

6.2.66 Digroup

Teorema 6.2.822. *Na teoria Digroup o resultado 6.1.12 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% PLM1
%% power-like functions
f(x) = x * g(x).
g(x) * x = x * g(x).
g(x * y) * x = x * g(y * x).
f(x) * g(x) = g(x) * f(x).
x * f(y * x) = f(x * y) * x.
f(x * x) = f(x) * f(x).
g(x) * g(x) = g(x * x).
f(x) * x = x * f(x).
%% Hypothesis of Lemma 1
f(x) * f(y) = f(y) * f(x).
g(x) * g(y) = g(y) * g(x).
end_of_list.
formulas(goals).
```

```
%% commutativity
x * y = y * x.
end_of_list.
```

□

Teorema 6.2.823. *Na teoria Digroup o resultado 6.1.98 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% hoop-eq
% Reading from file hp-ge-def.ax
x >= y <-> y * x = 0.
end_of_list.
formulas(goals).
% Reading from file hp-sum-lemma.gl
y >= x & x + x = x -> x + y = y.
end_of_list.
```

□

Lema 6.2.257. *Na teoria Digroup o resultado 6.1.99 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% theoremNNSNNSNN-eq-basic-lemmas
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x >= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x >= y cap x.
x >= x \ y.
x + y = x + (y \ x).
x + (z * y) >= z * (y * x).
z cap (y * x) >= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
% Reading from file theoremNNSNNSNN.gl
x'' * y'' = (x * y)''.
```

```
end_of_list.
```

□

Lema 6.2.258. *Na teoria Digroup o resultado 6.1.13 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% MNA
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x cap y)' = x nand y.
end_of_list.
```

□

Lema 6.2.259. *Na teoria Digroup o resultado 6.1.17 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% NNSSNN
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
```

```
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
x' = x' * (x * x'').
end_of_list.
```

□

Lema 6.2.260. *Na teoria Digroup o resultado 6.1.18 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% SNNNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
(x * x'')' = 1.
end_of_list.
```

□

Lema 6.2.261. *Na teoria Digroup o resultado 6.1.19 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
op(500,infix,"cup").
op(500,infix,"cap").
op(500,infix,"\").
op(500,infix,"nand").

formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
```

```
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% SSNNSNO
x cup y = x + (y * x).
x cap y = x * (x * y).
x \ y = (x + y) * y.
x' = 1 * x.
x nand y = x' + (x * y).
x <= y <-> y * x = 0.
% Reading from file lemma-basic.ax
x <= y cap x.
x <= x \ y.
x + y = x + (y \ x).
x + (z * y) <= z * (y * x).
z cap (y * x) <= (z cap y) * (z cap x).
x * (x cap y) = x * y.
end_of_list.
formulas(goals).
((x * y) * (x'' * y))' = 1.
end_of_list.
```

□

Lema 6.2.262. *Na teoria Digroup o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.263. *Na teoria Digroup o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Digroup
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
x + x' = 1 & x' * x = 1.
% PA2022_2
% conjugacy
```

```
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.67 Bilattice

Teorema 6.2.824. *Na teoria Bilattice o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Bilattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

-(x v y) = -x ^ -y.
--x = x.

-(x + y) = -x + -y.
-(x * y) = -x * -y.
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.825. *Na teoria Bilattice o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Bilattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
```

```
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

-(x v y) = -x ^ -y.
--x = x.

-(x + y) = -x + -y.
-(x * y) = -x * -y.
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) = UpMe(a,x2,z2) -> UpMe(x2,y2,z2) =
LoMe(x2,y2,z2))).
end_of_list.
```

□

6.2.68 Boolean monoid

Teorema 6.2.826. *Na teoria Boolean monoid o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Boolean monoid
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

(x * y) * z = x * (y * z).
e * x = x.
x * e = x.

((x v y) * z = (x * z) v (y * z)) & (x * (y v z) = (x * y) v (x * z)).

(0 * x = 0) & (x * 0 = 0).
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.827. *Na teoria Boolean monoid o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Boolean monoid
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

(x * y) * z = x * (y * z).
e * x = x.
x * e = x.

((x v y) * z = (x * z) v (y * z)) & (x * (y v z) = (x * y) v (x * z)).

(0 * x = 0) & (x * 0 = 0).
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) = (((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) = UpMe(a,x2,z2) -> UpMe(x2,y2,z2) = LoMe(x2,y2,z2))).
end_of_list.
```

□

Teorema 6.2.828. *Na teoria Boolean monoid o resultado 6.1.102 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Boolean monoid
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

(x * y) * z = x * (y * z).
e * x = x.
x * e = x.

((x v y) * z = (x * z) v (y * z)) & (x * (y v z) = (x * y) v (x * z)).

(0 * x = 0) & (x * 0 = 0).
```

```
% RL_3
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) = (((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> (UpMe(a,x2,y2) = UpMe(a,y2,z2) -> UpJo(x2,y2,z2) = LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.2.69 Commutative lattice-ordered monoids

Teorema 6.2.829. *Na teoria Commutative lattice-ordered monoids o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative lattice-ordered monoids
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).

x * y = y * x.
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

6.2.70 Commutative lattice-ordered rings

Teorema 6.2.830. *Na teoria Commutative lattice-ordered rings o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative lattice-ordered rings
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
```

```
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
(x < y) -> (x + z) < (y + z).
((0 < x) & (0 < y)) -> (0 < x * y).

x * y = y * x.
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.831. *Na teoria Commutative lattice-ordered rings o resultado 6.1.102 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative lattice-ordered rings
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
(x < y) -> (x + z) < (y + z).
((0 < x) & (0 < y)) -> (0 < x * y).

x * y = y * x.
% RL_3
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> (UpMe(a,x2,y2) = UpMe(a,y2,z2) -> UpJo(x2,y2,z2) = LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.2.71 Commutative lattice-ordered semigroups

Teorema 6.2.832. *Na teoria Commutative lattice-ordered semigroups o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative lattice-ordered semigroups
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
x * y = y * x.
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.833. *Na teoria Commutative lattice-ordered semigroups o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative lattice-ordered semigroups
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
x * y = y * x.
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> UpMe(x2,y2,z2) = LoMe(x2,y2,z2))).
end_of_list.
```

□

6.2.72 De Morgan monoids

Teorema 6.2.834. *Na teoria De Morgan monoids o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% De Morgan monoids
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * y = y * x.

x * y < z <-> y < (z' * x)'.

x < x * x.
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.835. *Na teoria De Morgan monoids o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% De Morgan monoids
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * y = y * x.

x * y < z <-> y < (z' * x)'.

x < x * x.
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
```

```
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) = UpMe(a,x2,z2) -> UpMe(x2,y2,z2) =
LoMe(x2,y2,z2))).
end_of_list.
```

□

Teorema 6.2.836. *Na teoria De Morgan monoids o resultado 6.1.102 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% De Morgan monoids
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * y = y * x.

x * y < z <-> y < (z' * x)'.

x < x * x.
% RL_3
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> (UpMe(a,x2,y2) = UpMe(a,y2,z2) -> UpJo(x2,y2,z2) =
LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.2.73 Distributive lattice ordered semigroups

Teorema 6.2.837. *Na teoria Distributive lattice ordered semigroups o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Distributive lattice ordered semigroups
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
```

```
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).

(x * y) * z = x * (y * z).

x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.838. *Na teoria Distributive lattice ordered semigroups o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Distributive lattice ordered semigroups
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).

(x * y) * z = x * (y * z).

x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> UpMe(x2,y2,z2) = LoMe(x2,y2,z2))).
end_of_list.
```

□

Teorema 6.2.839. *Na teoria Distributive lattice ordered semigroups o resultado 6.1.102 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Distributive lattice ordered semigroups
(x v y) v z = x v (y v z).
x v y = y v x.
```

```
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).

(x * y) * z = x * (y * z).

x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
% RL_3
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> (UpMe(a,x2,y2) = UpMe(a,y2,z2) -> UpJo(x2,y2,z2) = LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.2.74 F-rings

Teorema 6.2.840. *Na teoria F-rings o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% F-rings
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
(x < y) -> (x + z) < (y + z).
((0 < x) & (0 < y)) -> (0 < x * y).
(x ^ y = 0 & z > 0) -> ((x * z) ^ y = 0 & (z * x) ^ y = 0).
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.841. *Na teoria F-rings o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% F-rings
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
(x < y) -> (x + z) < (y + z).
((0 < x) & (0 < y)) -> (0 < x * y).
(x ^ y = 0 & z > 0) -> ((x * z) ^ y = 0 & (z * x) ^ y = 0).
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) = UpMe(a,x2,z2) -> UpMe(x2,y2,z2) =
LoMe(x2,y2,z2))).
end_of_list.
```

□

Teorema 6.2.842. *Na teoria F-rings o resultado 6.1.102 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% F-rings
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
(x < y) -> (x + z) < (y + z).
((0 < x) & (0 < y)) -> (0 < x * y).
(x ^ y = 0 & z > 0) -> ((x * z) ^ y = 0 & (z * x) ^ y = 0).
% RL_3
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
```

```
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> (UpMe(a,x2,y2) = UpMe(a,y2,z2) -> UpJo(x2,y2,z2) = LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.2.75 Lattice-ordered rings

Teorema 6.2.843. *Na teoria Lattice-ordered rings o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Lattice-ordered rings
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).

(x < y) -> (x + z) < (y + z).

((0 < x) & (0 < y)) -> (0 < x * y).
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.844. *Na teoria Lattice-ordered rings o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Lattice-ordered rings
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
```

```
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).

(x < y) -> (x + z) < (y + z).

((0 < x) & (0 < y)) -> (0 < x * y).
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) = UpMe(a,x2,z2) -> UpMe(x2,y2,z2)
= LoMe(x2,y2,z2))).
end_of_list.
```

□

Teorema 6.2.845. *Na teoria Lattice-ordered rings o resultado 6.1.102 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Lattice-ordered rings
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).

(x < y) -> (x + z) < (y + z).

((0 < x) & (0 < y)) -> (0 < x * y).
% RL_3
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> (UpMe(a,x2,y2) = UpMe(a,y2,z2) -> UpJo(x2,y2,z2) = LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.2.76 Lattice-ordered semigroups

Teorema 6.2.846. *Na teoria Lattice-ordered semigroups o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Lattice-ordered semigroups
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.847. *Na teoria Lattice-ordered semigroups o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Lattice-ordered semigroups
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) =
(((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> UpMe(x2,y2,z2) = LoMe(x2,y2,z2))).
end_of_list.
```

□

6.2.77 Multiplicative lattices

Teorema 6.2.848. *Na teoria Multiplicative lattices o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Multiplicative lattices
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.849. *Na teoria Multiplicative lattices o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Multiplicative lattices
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) = (((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> UpMe(x2,y2,z2) = LoMe(x2,y2,z2))).
end_of_list.
```

□

6.2.78 Normal valued lattice-ordered groups

Teorema 6.2.850. *Na teoria Normal valued lattice-ordered groups o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal valued lattice-ordered groups
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.
x < y -> ((x * z < y * z) & (z * x < z * y)).

((x v x') * (y v y')) < (((y v y') * (y v y')) * ((x v x') * (x v x'))).
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) =
LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.851. *Na teoria Normal valued lattice-ordered groups o resultado 6.1.102 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Normal valued lattice-ordered groups
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.
x < y -> ((x * z < y * z) & (z * x < z * y)).

((x v x') * (y v y')) < (((y v y') * (y v y')) * ((x v x') * (x v x'))).
% RL_3
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) = (((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
```

```
UpMe(a,x2,z2) -> (UpMe(a,x2,y2) = UpMe(a,y2,z2) -> UpJo(x2,y2,z2) = LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.2.79 Sequential algebras

Teorema 6.2.852. *Na teoria Sequential algebras o resultado 6.1.100 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Sequential algebras
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

(x @ y) @ z = x @ (y @ z).
e @ x = x.
x @ e = x.

(x @ y) ^ z = 0 <-> (x \ z) ^ y = 0.

(x @ y) ^ z = 0 <-> (z / y) ^ x = 0.

x \ e = e / x.

x * (y / z) < (x * y) / z.
% RL_1
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
end_of_list.
formulas(goals).
(all x1 all y1 all w UpMe(a ^ x1,y1,w) v (y1 ^ w) =
(((a ^ x1) ^ y1) v w) ^ (((a ^ x1) ^ w) v y1)) -> UpMe(a ^ z1,z2,z3) = LoMe(a ^ z1,z2,z3).
end_of_list.
```

□

Teorema 6.2.853. *Na teoria Sequential algebras o resultado 6.1.101 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Sequential algebras
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
```

```
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

(x @ y) @ z = x @ (y @ z).
e @ x = x.
x @ e = x.

(x @ y) ^ z = 0 <-> (x \ z) ^ y = 0.

(x @ y) ^ z = 0 <-> (z / y) ^ x = 0.

x \ e = e / x.

x * (y / z) < (x * y) / z.
% RL_2
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) = (((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> UpMe(x2,y2,z2) = LoMe(x2,y2,z2))).
end_of_list.
```

□

Teorema 6.2.854. *Na teoria Sequential algebras o resultado 6.1.102 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Sequential algebras
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

(x @ y) @ z = x @ (y @ z).
e @ x = x.
x @ e = x.

(x @ y) ^ z = 0 <-> (x \ z) ^ y = 0.

(x @ y) ^ z = 0 <-> (z / y) ^ x = 0.

x \ e = e / x.

x * (y / z) < (x * y) / z.
% RL_3
UpMe(x,y,z) = x ^ (y v z).
LoMe(x,y,z) = (x ^ y) v (x ^ z).
UpJo(x,y,z) = (x v y) ^ (x v z).
LoJo(x,y,z) = x v (y ^ z).
UpMe(a ^ x1,y1,z1) v (y1 ^ z1) = (((a ^ x1) ^ y1) v z1) ^ (((a ^ x1) ^ z1) v y1).
UpMe(x,y,z) = UpMe(x,y,a ^ z) v UpMe(x,z,a ^ y).
end_of_list.
formulas(goals).
```

```
(all x2 all y2 all z2 (UpMe(a,x2,y2) =
UpMe(a,x2,z2) -> (UpMe(a,x2,y2) = UpMe(a,y2,z2) -> UpJo(x2,y2,z2) = LoJo(x2,y2,z2)))).
end_of_list.
```

□

6.2.80 Commutative integral ordered monoid

Teorema 6.2.855. *Na teoria Commutative integral ordered monoid o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative integral ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
x < y | y < x.
x < y -> x * z < y * z.
x * y = y * x.
x < 1.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.856. *Na teoria Commutative integral ordered monoid o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative integral ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
x < y | y < x.
x < y -> x * z < y * z.
x * y = y * x.
x < 1.
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.857. *Na teoria Commutative integral ordered monoid o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative integral ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
x < y | y < x.
x < y -> x * z < y * z.
x * y = y * x.
x < 1.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.858. *Na teoria Commutative integral ordered monoid o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative integral ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
x < y | y < x.
x < y -> x * z < y * z.
x * y = y * x.
x < 1.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.859. *Na teoria Commutative integral ordered monoid o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative integral ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
x < y | y < x.
x < y -> x * z < y * z.
x * y = y * x.
x < 1.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
```

```
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.81 Commutative ordered monoids

Teorema 6.2.860. *Na teoria Commutative ordered monoids o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x * y = y * x.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.861. *Na teoria Commutative ordered monoids o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x * y = y * x.
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.862. *Na teoria Commutative ordered monoids o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x * y = y * x.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.863. *Na teoria Commutative ordered monoids o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x * y = y * x.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.864. *Na teoria Commutative ordered monoids o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
```

```
x * y = y * x.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.82 Commutative ordered semigroups

Teorema 6.2.865. *Na teoria Commutative ordered semigroups o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative ordered semigroups
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x * y = y * x.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.866. *Na teoria Commutative ordered semigroups o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative ordered semigroups
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x * y = y * x.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.83 Commutative partially ordered monoids

Teorema 6.2.867. *Na teoria Commutative partially ordered monoids o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative partially ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).

x * y = y * x.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.868. *Na teoria Commutative partially ordered monoids o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative partially ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).

x * y = y * x.
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.869. *Na teoria Commutative partially ordered monoids o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative partially ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).

x * y = y * x.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.870. *Na teoria Commutative partially ordered monoids o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative partially ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).

x * y = y * x.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.871. *Na teoria Commutative partially ordered monoids o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative partially ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).

x * y = y * x.
% HT_L5_2_E
```

```
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.84 Commutative partially ordered semigroups

Teorema 6.2.872. *Na teoria Commutative partially ordered semigroups o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative partially ordered semigroups

(x * y) * z = x * (y * z).

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

x * y = y * x.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.873. *Na teoria Commutative partially ordered semigroups o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative partially ordered semigroups

(x * y) * z = x * (y * z).

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

x * y = y * x.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.85 Commutative regular rings

Teorema 6.2.874. *Na teoria Commutative regular rings o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative regular rings
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

x * y = y * x.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

6.2.86 Commutative residuated partially ordered monoids

Teorema 6.2.875. *Na teoria Commutative residuated partially ordered monoids o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).

x * y = y * x.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.876. *Na teoria Commutative residuated partially ordered monoids o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).

x * y = y * x.
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.877. *Na teoria Commutative residuated partially ordered monoids o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).

x * y = y * x.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.878. *Na teoria Commutative residuated partially ordered monoids o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative residuated partially ordered monoids
x < x.
```

```
x < y & y < z -> x < z.
x < y & y < x -> x = y.
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).

x * y = y * x.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.879. *Na teoria Commutative residuated partially ordered monoids o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Commutative residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).

x * y = y * x.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.87 Integral ordered monoids

Teorema 6.2.880. *Na teoria Integral ordered monoids o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Integral ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
```

```
x < 1.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.881. *Na teoria Integral ordered monoids o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Integral ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x < 1.
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.882. *Na teoria Integral ordered monoids o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Integral ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x < 1.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.883. *Na teoria Integral ordered monoids o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Integral ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x < 1.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.884. *Na teoria Integral ordered monoids o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Integral ordered monoids
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

x < 1.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.88 Ordered monoids with zero

Teorema 6.2.885. *Na teoria Ordered monoids with zero o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered monoids with zero
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
```

```
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * 0 = 0 & 0 * x = 0.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.886. *Na teoria Ordered monoids with zero o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered monoids with zero
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * 0 = 0 & 0 * x = 0.
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.887. *Na teoria Ordered monoids with zero o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered monoids with zero
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * 0 = 0 & 0 * x = 0.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.888. *Na teoria Ordered monoids with zero o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered monoids with zero
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * 0 = 0 & 0 * x = 0.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.889. *Na teoria Ordered monoids with zero o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered monoids with zero
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * 0 = 0 & 0 * x = 0.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.89 Ordered semigroup

Teorema 6.2.890. *Na teoria Ordered semigroup o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered semigroup
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
```

```
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.891. *Na teoria Ordered semigroup o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered semigroup
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.90 Ordered semilattice

Teorema 6.2.892. *Na teoria Ordered semilattice o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered semilattice
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * y = y * x.
x * x = x.
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.893. *Na teoria Ordered semilattice o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered semilattice
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * y = y * x.
x * x = x.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.894. *Na teoria Ordered semilattice o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered semilattice
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * y = y * x.
x * x = x.
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.895. *Na teoria Ordered semilattice o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Ordered semilattice
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
x * y = y * x.
x * x = x.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
```

```
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.91 Partially ordered groups

Teorema 6.2.896. *Na teoria Partially ordered groups o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered groups

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.897. *Na teoria Partially ordered groups o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered groups

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.898. *Na teoria Partially ordered groups o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered groups

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.899. *Na teoria Partially ordered groups o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered groups

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.900. *Na teoria Partially ordered groups o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered groups

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

Lema 6.2.264. *Na teoria Partially ordered groups o resultado 6.1.10 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered groups

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

% PA2022_1
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 2.
(z * (x * z')) * (z * (x * z')) = z * ((x * x) * z').
end_of_list.
```

□

Lema 6.2.265. *Na teoria Partially ordered groups o resultado 6.1.11 e verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered groups

(x * y) * z = x * (y * z).
x * 1 = x.
```

```
1 * x = x.
x * x' = 1.
x' * x = 1.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

% PA2022_2
% conjugacy
all x all y exists z' (z * x) * z' = y.
end_of_list.
formulas(goals).
% n = 3.
(z * (x * z')) * ((z * (x * z')) * (z * (x * z'))) = z * (((x * x) * x) * z').
end_of_list.
```

□

6.2.92 Partially ordered monoids

Teorema 6.2.901. *Na teoria Partially ordered monoids o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.902. *Na teoria Partially ordered monoids o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
```

```
x < y -> ((x * z < y * z) & (z * x < z * y)).
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.903. *Na teoria Partially ordered monoids o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.904. *Na teoria Partially ordered monoids o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.905. *Na teoria Partially ordered monoids o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.93 Partially ordered semigroups

Teorema 6.2.906. *Na teoria Partially ordered semigroups o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered semigroups

(x * y) * z = x * (y * z).

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.907. *Na teoria Partially ordered semigroups o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Partially ordered semigroups

(x * y) * z = x * (y * z).
```

```
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.94 Residuated partially ordered monoids

Teorema 6.2.908. *Na teoria Residuated partially ordered monoids o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.

(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.909. *Na teoria Residuated partially ordered monoids o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.

(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
```

```
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.910. *Na teoria Residuated partially ordered monoids o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.

(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.911. *Na teoria Residuated partially ordered monoids o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.

(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).
% HT_L5_2_D
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.912. *Na teoria Residuated partially ordered monoids o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Residuated partially ordered monoids
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.

(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

6.2.95 Totally ordered monoids

Teorema 6.2.913. *Na teoria Totally ordered monoids o resultado 6.1.103 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Totally ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

x < y | y < x.
% HT_L5_2_A
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,1,z) <-> x < z.
end_of_list.
```

□

Teorema 6.2.914. *Na teoria Totally ordered monoids o resultado 6.1.104 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Totally ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
```

```
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

x < y | y < x.
% HT_L5_2_B
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 < y1.
end_of_list.
```

□

Teorema 6.2.915. *Na teoria Totally ordered monoids o resultado 6.1.105 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Totally ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

x < y | y < x.
% HT_L5_2_C
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
end_of_list.
```

□

Teorema 6.2.916. *Na teoria Totally ordered monoids o resultado 6.1.106 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Totally ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

x < y | y < x.
% HT_L5_2_D
```

```
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
hoa(x,y1*y2,z) <-> (exists u (hoa(x,y1,u) & hoa(u,y2,z))).
end_of_list.
```

□

Teorema 6.2.917. *Na teoria Totally ordered monoids o resultado 6.1.107 é verdadeiro.*

Demonstração. Pode ser obtida a partir da entrada em Prover9:

```
formulas(assumptions).
% Totally ordered monoids

(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

x < y -> ((x * z < y * z) & (z * x < z * y)).

x < y | y < x.
% HT_L5_2_E
% Hoare triple
hoa(x,y,z) <-> x*y < z.
end_of_list.
formulas(goals).
x < u & hoa(u,y1,w) & w < z -> hoa(x,y1,z).
end_of_list.
```

□

Capítulo 7

Conclusões e trabalho futuro

O objetivo era construir um sistema que autonomamente gerasse matemática com significado, nova e inteligível. A ideia foi usar conceitos e teoremas existentes. Esse resultado foi totalmente conseguido com a geração de novos teoremas, em áreas muito diferentes da álgebra, muitos deles com provas complexas e longas, e que as pessoas reconhecem. Uns teoremas são exportações para outras áreas de teoremas clássicos. A maioria são tipos de teoremas.

O trabalho futuro tem várias componentes:

1. Aumentar a base de dados de teorias e teoremas;

2. Aumentar a facilidade com que os utilizadores introduzem no sistema tipos de teoremas (a introdução de teoremas é fácil);

3. Usar técnicas de linguagem natural para extrair teoremas e axiomáticas automaticamente do arXiv;

4. Criar o generalizador de teoremas (ie, sempre que o mesmo teorema é válido em duas áreas diferentes da matemática, o sistema tenta encontrar uma teoria mais geral que contenha as duas áreas e onde seja possível provar aquele teorema);

5. Integrar todo o sistema com o ProverX (Yves Robert) e restantes ferramentas já produzidas[1], tais como Forbidden Substructure Theorem (Fernando Ferreira), SGV (Jorge Raminhos), etc.

[1] `https://docentes.fct.unl.pt/jj-araujo/software`

Capítulo 8

Anexos

São apresentadas as base de dados de modelos e de teorias. Por último, é anexado um artigo que descreve um package para GAP.

8.1 Base de dados de modelos

São mostrados de seguida modelos de ordem 7 da base de dados (para modelos de outras ordens, consultar `www.marciex.com`).

Modelo 8.8.1 (Model #93).

*	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	6
2	2	3	4	5	6	6	6
3	3	4	5	6	6	6	6
4	4	5	6	6	6	6	6
5	5	6	6	6	6	6	6
6	6	6	6	6	6	6	6

https://marciedb.pythonanywhere.com/model/93
https://arxiv.org/pdf/0709.0522.pdf#page=7

Modelo 8.8.2 (Model #94).

*	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1
2	0	1	2	2	2	2	2
3	0	1	2	3	3	3	3
4	0	1	2	3	4	4	4
5	0	1	2	3	4	5	5
6	0	1	2	3	4	5	6

https://marciedb.pythonanywhere.com/model/94
https://arxiv.org/pdf/0709.0522.pdf#page=7

Modelo 8.8.3 (Model #98).

*	0	1	2	3	4	5	6
0	0	0	0	3	3	3	6
1	0	0	1	3	3	4	6
2	0	0	2	3	3	5	6
3	0	0	3	3	3	6	6
4	0	1	3	3	4	6	6
5	0	2	3	3	5	6	6
6	0	3	3	3	6	6	6

https://marciedb.pythonanywhere.com/model/98
https://arxiv.org/pdf/0802.1859.pdf#page=26

Modelo 8.8.4 (Model #113).

*	0	1	2	3	4	5	6
0	6	6	6	6	6	6	6
1	4	6	6	4	4	6	6
2	4	5	6	4	4	6	6
3	2	2	2	6	6	6	6
4	2	2	2	5	6	6	6
5	0	2	2	4	4	6	6
6	0	1	2	3	4	5	6

https://marciedb.pythonanywhere.com/model/113
https://arxiv.org/pdf/0901.1630.pdf#page=14

Modelo 8.8.5 (Model #114).

*	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	1	0	0	1	1
2	0	1	1	0	0	1	2
3	0	0	0	3	3	3	3
4	0	0	0	3	3	3	4
5	0	1	1	3	3	5	5
6	0	1	2	3	4	5	6

https://marciedb.pythonanywhere.com/model/114
https://arxiv.org/pdf/0901.1630.pdf#page=14

Modelo 8.8.6 (Model #139).

*	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	0	4	5	6	3
2	2	0	1	6	3	4	5
3	3	4	5	0	6	2	1
4	4	5	6	1	0	3	2
5	5	6	3	2	1	0	4
6	6	3	4	5	2	1	0

https://marciedb.pythonanywhere.com/model/139
https://arxiv.org/pdf/0909.1065.pdf#page=21

Modelo 8.8.7 (Model #140).

*	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	0	4	3	6	5
2	2	0	3	5	6	1	4
3	3	4	5	6	1	0	2
4	4	3	6	1	5	2	0
5	5	6	1	0	2	4	3
6	6	5	4	2	0	3	1

https://marciedb.pythonanywhere.com/model/140
https://arxiv.org/pdf/0909.1065.pdf#page=21

Modelo 8.8.8 (Model #141).

*	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	1	5	6	0	4
3	3	5	4	6	0	2	1
4	4	5	6	0	3	1	2
5	5	6	0	1	2	4	3
6	6	0	4	2	1	3	5

https://marciedb.pythonanywhere.com/model/141
https://arxiv.org/pdf/0909.1065.pdf#page=22

Modelo 8.8.9 (Model #142).

*	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	1	5	6	0	4
3	3	4	5	6	0	1	2
4	4	5	6	0	3	2	1
5	5	6	0	2	1	4	3
6	6	0	4	1	2	3	5

https://marciedb.pythonanywhere.com/model/142
https://arxiv.org/pdf/0909.1065.pdf#page=22

Modelo 8.8.10 (Model #161).

*	0	1	2	3	4	5	6
0	0	2	3	1	6	4	5
1	6	1	4	0	5	2	3
2	5	0	2	6	3	1	4
3	4	5	0	3	2	6	1
4	3	6	1	5	4	0	2
5	2	3	6	4	1	5	0
6	1	4	5	2	0	3	6

https://marciedb.pythonanywhere.com/model/161
https://arxiv.org/pdf/1509.00796.pdf#page=5

Modelo 8.8.11 (Model #165).

*	0	1	2	3	4	5	6
0	1	2	0	5	4	6	3
1	0	3	6	4	1	2	5
2	5	0	4	2	3	1	6
3	3	1	5	6	2	4	0
4	6	5	2	1	0	3	4
5	2	4	3	0	6	5	1
6	4	6	1	3	5	0	2

https://marciedb.pythonanywhere.com/model/165
https://arxiv.org/pdf/1509.00796.pdf#page=5

Modelo 8.8.12 (Model #199).

*	0	1	2	3	4	5	6
0	0	0	0	0	4	4	0
1	0	0	0	0	4	4	0
2	0	0	0	0	4	4	0
3	0	0	0	0	4	4	0
4	4	4	4	4	4	4	4
5	4	4	4	4	4	4	4
6	0	0	2	3	4	5	6

https://marciedb.pythonanywhere.com/model/199
https://arxiv.org/pdf/1503.00915.pdf#page=3

Modelo 8.8.13 (Model #307).

*	0	1	2	3	4	5	6
0	0	4	1	5	2	6	3
1	5	2	6	3	0	4	1
2	3	0	4	1	5	2	6
3	1	5	2	6	3	0	4
4	6	3	0	4	1	5	2
5	4	1	5	2	6	3	0
6	2	6	3	0	4	1	5

https://marciedb.pythonanywhere.com/model/307
https://arxiv.org/pdf/1403.2564.pdf#page=7

Modelo 8.8.14 (Model #419).

*	0	1	2	3	4	5	6
0	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3
2	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3
4	3	3	3	3	3	3	3
5	3	3	3	3	3	3	3
6	0	1	2	3	4	5	3

https://marciedb.pythonanywhere.com/model/419
https://arxiv.org/pdf/1101.0949.pdf#page=10

Modelo 8.8.15 (Model #431).

*	0	1	2	3	4	5	6
0	0	3	4	3	4	5	3
1	6	1	2	6	5	5	6
2	2	2	2	2	2	2	2
3	3	3	5	3	5	5	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	5	6	5	5	6

https://marciedb.pythonanywhere.com/model/431
https://arxiv.org/pdf/1102.2127.pdf#page=15

Modelo 8.8.16 (Model #437).

*	0	1	2	3	4	5	6
0	0	0	1	2	2	2	4
1	0	2	2	3	4	4	6
2	1	2	3	3	4	5	6
3	2	3	3	4	5	6	6
4	2	4	4	5	5	6	6
5	2	4	5	6	6	6	6
6	4	6	6	6	6	6	6

https://marciedb.pythonanywhere.com/model/437
https://arxiv.org/pdf/1103.0795.pdf#page=2
https://arxiv.org/pdf/1202.1337.pdf#page=2

Modelo 8.8.17 (Model #438).

*	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	0	5	6	3	4
2	2	0	1	6	5	4	3
3	3	5	6	4	0	1	2
4	4	6	5	0	3	2	1
5	5	3	4	1	2	6	0
6	6	4	3	2	1	0	5

https://marciedb.pythonanywhere.com/model/438
https://arxiv.org/pdf/1103.0054.pdf#page=4

Modelo 8.8.18 (Model #498).

*	0	1	2	3	4	5	6
0	0	4	1	5	2	6	3
1	4	1	5	2	6	3	0
2	1	5	2	6	3	0	4
3	5	2	6	3	0	4	1
4	2	6	3	0	4	1	5
5	6	3	0	4	1	5	2
6	3	0	4	1	5	2	6

https://marciedb.pythonanywhere.com/model/498
https://arxiv.org/pdf/1110.1849.pdf#page=4

Modelo 8.8.19 (Model #549).

*	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	2	0	0
2	0	0	0	0	0	2	3
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0

https://marciedb.pythonanywhere.com/model/549
https://arxiv.org/pdf/1210.1475.pdf#page=1

Modelo 8.8.20 (Model #611).

*	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	0	1	2	3	4	5	6
2	0	1	2	3	4	5	6
3	0	2	1	5	6	3	4
4	0	2	1	5	6	3	4
5	0	2	1	5	6	3	4
6	0	2	1	5	6	3	4

https://marciedb.pythonanywhere.com/model/611
https://arxiv.org/pdf/1307.5404.pdf#page=3

8.2 Base de dados de teorias

A codificação para Prover9 foi obtida a partir de teorias disponibilizadas pelo Professor Peter Jipsen (Chapman University) [20] e a partir de artigos do arXiv, ResearchGate, etc. Mais pormenores em www.marciex.com.

Teoria 8.2.1 (#1 Almost distributive lattices)**.**

```
formulas(assumptions).
% neardistributive lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = x ^ (y v (x ^ (z v (x ^ y)))).
x v (y ^ z) = x v (y ^ (x v (z ^ (x v y)))).

%
w ^ (u v (x ^ (y v (x ^ z)))) < u v ((x ^ (y v (x ^ z))) ^ ((w v (x ^ y)) v (x ^ z))).
w v (u ^ (x v (y ^ (x v z)))) > u ^ ((x v (y ^ (x v z))) v ((w ^ (x v y)) ^ (x v z))).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/1

Teoria 8.2.2 (#2 BCI-algebras)**.**

```
formulas(assumptions).
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/2

Teoria 8.2.3 (#3 BCK-algebras)**.**

```
formulas(assumptions).
% BCI algebra
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
%
x * 0 = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/3

Teoria 8.2.4 (#4 BCK-join-semilattice)**.**

```
formulas(assumptions).
(x @ y) @ ((y @ z) @ (x @ z)) = 1.
1 @ x = x.
x @ 1 = 1.
x @ (x v y) = 1.
x v ((x @ y) @ y) = ((x @ y) @ y).
x v x = x.
x v y = y v x.
(x v y) v z = x v (y v z).

x < y <-> x @ y = 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/4

Teoria 8.2.5 (#5 BCK-lattices)**.**

```
formulas(assumptions).
% BCK-join-semilattice
(x @ y) @ ((y @ z) @ (x @ z)) = 1.
1 @ x = x.
x @ 1 = 1.
x @ (x v y) = 1.
x v ((x @ y) @ y) = ((x @ y) @ y).
x v x = x.
x v y = y v x.
(x v y) v z = x v (y v z).
% BCK-meet-semilattice
% (x @ y) @ ((y @ z) @ (x @ z)) = 1.
% 1 @ x = x.
% x @ 1 = 1.
(x ^ y) @ y = 1.
x ^ ((x @ y) @ y) = x.
x ^ x = x.
x ^ y = y ^ x.
(x ^ y) ^ z = x ^ (y ^ z).

x < y <-> x @ y = 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/5

Teoria 8.2.6 (#6 BCK-meet-semilattice)**.**

```
formulas(assumptions).
(x @ y) @ ((y @ z) @ (x @ z)) = 1.
1 @ x = x.
x @ 1 = 1.
(x ^ y) @ y = 1.
x ^ ((x @ y) @ y) = x.
x ^ x = x.
x ^ y = y ^ x.
(x ^ y) ^ z = x ^ (y ^ z).

x < y <-> x @ y = 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/6

Teoria 8.2.7 (#7 Bilattice).

```
formulas(assumptions).
% v ^ lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

% + * lattice
(x + y) + z = x + (y + z).
x + y = y + x.
(x * y) * z = x * (y * z).
x * y = y * x.
(x + y) * x = x.
(x * y) + x = x.

% De Morgan
-(x v y) = -x ^ -y.
--x = x.

% commutes
-(x + y) = -x + -y.
-(x * y) = -x * -y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/7

Teoria 8.2.8 (#8 BL-algebras).

```
formulas(assumptions).
% bounded lattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
% commutative monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * y = y * x.

% residual
x * y < z <-> y < x @ z.
% prelinearity
(x @ y) v (y @ x) = 1.
% BL
x * (x @ y) = x ^ y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/8

Teoria 8.2.9 (#9 Boolean algebra).

```
formulas(assumptions).
% bounded distributive lattice
```

```
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
% complement
x ^ -x = 0.
x v -x = 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/9

Teoria 8.2.10 (#10 Boolean monoid)**.**

```
formulas(assumptions).
% boolean algebra
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

% monoid
(x * y) * z = x * (y * z).
e * x = x.
x * e = x.

% join-preserving
((x v y) * z = (x * z) v (y * z)) & (x * (y v z) = (x * y) v (x * z)).

% normal
(0 * x = 0) & (x * 0 = 0).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/10

Teoria 8.2.11 (#11 Boolean ring)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x * y = y * x.
x * 0 = x.
0 * x = x.
x * x = 0.
(x @ y) @ z = x @ (y @ z).
1 @ x = x.
x @ 1 = x.
x @ x = x.
x @ (y * z) = (x @ y) * (x @ z).
(x * y) @ z = (x @ z) * (y @ z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/11

Teoria 8.2.12 (#12 Bounded lattice)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
% bounded
0 v x = x.
1 v x = 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/12

Teoria 8.2.13 (#13 Bounded residuated lattices)**.**

```
formulas(assumptions).
% monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.

% lattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

% @ left residual of *
(y < x @ z) <-> (x * y < z).
% / right residual
(x < z / y) <-> (x * y < z).

% bounded
exists a (a v x = x).
exists b (b v x = b).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/13

Teoria 8.2.14 (#14 Brouwerian algebras)**.**

```
formulas(assumptions).
% BL-algebra
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
(x * y) * z = x * (y * z).
x * 1 = x.
```

```
% 1 * x = x.
x * y = y * x.
x * y < z <-> y < x @ z.
(x @ y) v (y @ x) = 1.
x * (x @ y) = x ^ y.
%
x ^ y = x * y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/14

Teoria 8.2.15 (#15 Brouwerian semilattices)**.**

```
formulas(assumptions).
% hoop
x * y = y * x.
(x * y) * z = x * (y * z).
1 * x = x.
x @ (y @ z) = (x * y) @ z.
x @ x = 1.
(x @ y) * x = (y @ x) * y.
% idempotent
x * x = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/15

Teoria 8.2.16 (#16 Chains)**.**

```
formulas(assumptions).
% partially ordered set
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

% total order
x < y | y < x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/16

Teoria 8.2.17 (#17 Clifford algebras)**.**

```
formulas(assumptions).
% inverse semigroup
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
(x * x') * (y * y') = (y * y') * (x * x').

% completely regular semigroups
(x * x') * x = x.
(x' * x) * x' = x'.
x * x' = x' * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/17

Teoria 8.2.18 (#18 Commutative BCK-algebras)**.**

```
formulas(assumptions).
% BCK algebra
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
% commutative
x * (x * y) = y * (y * x).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/18

Teoria 8.2.19 (#19 Commutative integral ordered monoid)**.**

```
formulas(assumptions).
% commutative ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
% 1 * x = x.
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
x < y | y < x.
x < y -> x * z < y * z.
% x < y -> z * x < z * y.
x * y = y * x.
% integral
x < 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/19

Teoria 8.2.20 (#20 Commutative lattice-ordered monoids)**.**

```
formulas(assumptions).
% lattice-ordered monoid
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/20

Teoria 8.2.21 (#21 Commutative lattice-ordered rings)**.**

```
formulas(assumptions).
% lattice-ordered ring
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
(x < y) -> (x + z) < (y + z).
((0 < x) & (0 < y)) -> (0 < x * y).

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/21

Teoria 8.2.22 (#22 Commutative lattice-ordered semigroups)**.**

```
formulas(assumptions).
%  lattice-ordered semigroup
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/22

Teoria 8.2.23 (#23 Commutative ordered monoids)**.**

```
formulas(assumptions).
% totally ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/23

Teoria 8.2.24 (#24 Commutative ordered semigroups)**.**

```
formulas(assumptions).
% ordered semigroup
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/24

Teoria 8.2.25 (#25 Commutative partially ordered monoids)**.**

```
formulas(assumptions).
% partially ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/25

Teoria 8.2.26 (#26 Commutative partially ordered semigroups)**.**

```
formulas(assumptions).
% partially ordered semigroup
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/26

Teoria 8.2.27 (#27 Commutative regular rings)**.**

```
formulas(assumptions).
```

```
% regular ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
exists y (x * y) * x = x.

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/27

Teoria 8.2.28 (#28 Commutative residuated partially ordered monoids)**.**

```
formulas(assumptions).
% residuated partially ordered monoid
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/28

Teoria 8.2.29 (#29 Complemented distributive lattices)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

% bounded
0 v x = x.
1 v x = 1.

% distributive
x ^ (y v z) = (x ^ y) v (x ^ z).

% complemented
x v x' = 1.
x ^ x' = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/29

Teoria 8.2.30 (#30 Complemented lattices)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

% bounded
0 v x = x.
1 v x = 1.

% complemented
all x exists y (x v y = 1 & x ^ y = 0).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/30

Teoria 8.2.31 (#31 Complemented modular lattices)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
% bounded
0 v x = x.
1 v x = 1.
% complemented
all x exists y (x v y = 1 & x ^ y = 0).
% modular identity
((x ^ z) v y) ^ z = (x ^ z) v (y ^ z).

end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/31

Teoria 8.2.32 (#32 Loop)**.**

```
formulas(assumptions).
% quasigroup
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.

% identity
x * e = x.
x = e * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/32

Teoria 8.2.33 (#33 De Morgan monoids)**.**

```
formulas(assumptions).
% distributive lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).

% commutative monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * y = y * x.

% involutive residuated
x * y < z <-> y < (z' * x)'.

% square-increasing
x < x * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/33

Teoria 8.2.34 (#34 Dense linear orders)**.**

```
formulas(assumptions).
% chain
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
x < y | y < x.

% dense
x < y -> exists z (x < z & z < y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/34

Teoria 8.2.35 (#35 Directoids)**.**

```
formulas(assumptions).
x * x = x.
(x * y) * x = x * y.
y * (x * y) = x * y.
x * ((x * y) * z) = (x * y) * z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/35

Teoria 8.2.36 (#36 Distributive lattice ordered semigroups)**.**

```
formulas(assumptions).
% distributive lattice
(x v y) v z = x v (y v z).
```

```
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).

% semigroup
(x * y) * z = x * (y * z).

% distributive
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/36

Teoria 8.2.37 (#37 Division ring)**.**

```
formulas(assumptions).
% ring with identity
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

% non-trivial
0 != 1.

% every non-zero element has a multiplicative inverse
x != 0 -> exists y (x * y = 1).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/37

Teoria 8.2.38 (#39 F-rings)**.**

```
formulas(assumptions).
% lattice-ordered ring
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
(x < y) -> (x + z) < (y + z).
((0 < x) & (0 < y)) -> (0 < x * y).
%
(x ^ y = 0 & z > 0) -> ((x * z) ^ y = 0 & (z * x) ^ y = 0).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/39

Teoria 8.2.39 (#40 Fields)**.**

```
formulas(assumptions).
% commutative rings with identity
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

% non-trivial
0 != 1.

% every non-zero element has a multiplicative inverse
x != 0 -> exists y (x * y = 1).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/40

Teoria 8.2.40 (#41 FLcw-algebras)**.**

```
formulas(assumptions).
% bounded distributive lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
% residual of ^
x ^ y < z <-> y < x @ z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/41

Teoria 8.2.41 (#42 Function rings)**.**

```
formulas(assumptions).
% lattice-ordered ring
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
```

```
(x + y) * z = (x * z) + (y * z).
(x < y) -> (x + z) < (y + z).
((0 < x) & (0 < y)) -> (0 < x * y).
%
(x ^ y = 0 & z > 0) -> ((x * z) ^ y = 0 & (z * x) ^ y = 0).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/42

Teoria 8.2.42 (#43 Generalized Boolean algebras)**.**

```
formulas(assumptions).
% brouwerian algebra
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
(x * y) * z = x * (y * z).
x * 1 = x.
% 1 * x = x.
x * y = y * x.
x * y < z <-> y < x @ z.
(x @ y) v (y @ x) = 1.
x * (x @ y) = x ^ y.
x ^ y = x * y.
%
x v y = (x @ y) @ y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/43

Teoria 8.2.43 (#44 Hilbert algebras)**.**

```
formulas(assumptions).
x @ (y @ x) = 1.
(x @ (y @ z)) @ ((x @ y) @ (x @ z)) = 1.
((x @ y = 1) & (y @ x = 1)) -> (x = y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/44

Teoria 8.2.44 (#45 Hoop)**.**

```
formulas(assumptions).
% commutative monoid
x * y = y * x.
(x * y) * z = x * (y * z).
1 * x = x.
%
x @ (y @ z) = (x * y) @ z.
x @ x = 1.
(x @ y) * x = (y @ x) * y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/45

Teoria 8.2.45 (#46 Implication algebras)**.**

```
formulas(assumptions).
% Tarski algebra
(x @ y) @ x = x.
(x @ y) @ y = (y @ x) @ x.
x @ (y @ z) = y @ (x @ z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/46

Teoria 8.2.46 (#47 Implicative lattices)**.**

```
formulas(assumptions).
% distributive lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
x ^ (y v z) = (x ^ y) v (x ^ z).
%
x @ (y v z) = (x @ y) v (x @ z).
x @ (y ^ z) = (x @ y) ^ (x @ z).
(x v y) @ z = (x @ z) ^ (y @ z).
(x ^ y) @ z = (x @ z) v (y @ z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/47

Teoria 8.2.47 (#48 Integral domains)**.**

```
formulas(assumptions).
% commutative ring with identity
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * y = y * x.
1 * x = x.
x * 1 = x.

% has no zero divisors
(x * y = 0) -> (x = 0 | y = 0).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/48

Teoria 8.2.48 (#49 Integral ordered monoids)**.**

```
formulas(assumptions).
% ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.

% integral
x < 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/49

Teoria 8.2.49 (#50 Involutive lattices)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

% De Morgan involution
-(x ^ y) = -x v -y.
--x = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/50

Teoria 8.2.50 (#51 Join-semilattices)**.**

```
formulas(assumptions).
x < x.
(x < y & y < x) -> (x = y).
(x < y & y < z) -> (x < z).
exists w (x < w & y < w & ((x < z & y < z) -> (w < z))).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/51

Teoria 8.2.51 (#52 Lattice-ordered pregroups)**.**

```
formulas(assumptions).
% abelian semigroup
(x + y) + z = x + (y + z).
x + y = y + x.

% lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
```

```
(x v y) ^ x = x.
(x ^ y) v x = x.

--x = x.
0 + 0 = 0.
x + -x = 0.
(x < y) <-> (0 < -x + y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/52

Teoria 8.2.52 (#53 Lattice-ordered rings)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

% ring
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).

% order-preserving
(x < y) -> (x + z) < (y + z).

% 0 closed under *
((0 < x) & (0 < y)) -> (0 < x * y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/53

Teoria 8.2.53 (#54 Lattice-ordered semigroups)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
% semigroup
(x * y) * z = x * (y * z).
% distributes over v
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/54

Teoria 8.2.54 (#55 Lineales)**.**

```
formulas(assumptions).
% residuated partially ordered monoid
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).

% commutative
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/55

Teoria 8.2.55 (#56 Linear Heyting algebras)**.**

```
formulas(assumptions).
% Heyting algebras
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ y < z <-> y < x @ z.
%
(x @ y) v (y @ x) = 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/56

Teoria 8.2.56 (#57 Linear orders)**.**

```
formulas(assumptions).
% chain
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
x < y | y < x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/57

Teoria 8.2.57 (#58 M-zeroids)**.**

```
formulas(assumptions).
% commutative semigroup
(x + y) + z = x + (y + z).
x + y = y + x.
```

```
% lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

%
-x = x.
x + 0 = 0.
x + -x = 0.
x < y <-> 0 = -x + y.
x + (y v z) = (x + y) v (x + z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/58

Teoria 8.2.58 (#59 Medial quasigroups)**.**

```
formulas(assumptions).
% quasigroup
(x * y) / y = x.
(x / y) * y = x.
x \ (x * y) = y.
x * (x \ y) = y.

% medial
(x * y) * (z * w) = (x * z) * (y * w).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/59

Teoria 8.2.59 (#60 Meet-semilattices)**.**

```
formulas(assumptions).
x < x.
(x < y & y < x) -> (x = y).
(x < y & y < z) -> (x < z).
exists w (w < x & w < y & ((z < x & z < y) -> (z < w))).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/60

Teoria 8.2.60 (#61 Modular ortholattices)**.**

```
formulas(assumptions).
% ortholattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x v x' = 1.
x ^ x' = 0.
```

```
(x v y)' = x' ^ y'.
(x ^ y)' = x' v y'.

% modular law
(x < z) -> ((x v y) ^ z < x v (y ^ z)).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/61

Teoria 8.2.61 (#62 Monadic algebras)**.**

```
formulas(assumptions).
% boolean algebra
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

% unary closure operator
f(x v y) = f(x) v f(y).
f(0) = 0.
x < f(x) & f(x) = f(f(x)).

% self conjugated
f(x) ^ y = 0 <-> x ^ f(y) = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/62

Teoria 8.2.62 (#63 Quasigroup)**.**

```
formulas(assumptions).
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/63

Teoria 8.2.63 (#64 Moufang quasigroups)**.**

```
formulas(assumptions).
% quasigroup
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.

% moufang law
(y * e = y) -> (((x * y) * z) * x = x * (y * ((e * z) * x))).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/64

Teoria 8.2.64 (#65 Multiplicative lattices)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

%
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/65

Teoria 8.2.65 (#66 Multiplicative semilattices)**.**

```
formulas(assumptions).
% semilattice
(x v y) v z = x v (y v z).
x v y = y v x.
x v x = x.

%
x * (y v z) = (x * y) v (x * z).
(x v y) * z = (x * z) v (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/66

Teoria 8.2.66 (#67 Near-fields)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x != 0 -> (exists y (x * y) = 1).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/67

Teoria 8.2.67 (#68 Near-ring)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
```

```
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/68

Teoria 8.2.68 (#69 Near-rings with identity)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z).
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/69

Teoria 8.2.69 (#70 Neardistributive lattices)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.

%
x ^ (y v z) = x ^ (y v (x ^ (z v (x ^ y)))).
x v (y ^ z) = x v (y ^ (x v (z ^ (x v y)))).

end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/70

Teoria 8.2.70 (#71 Normal bands)**.**

```
formulas(assumptions).
% band
(x * y) * z = x * (y * z).
x * x = x.

% normal
((x * y) * z) * x = ((x * z) * y) * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/71

Teoria 8.2.71 (#72 Normal valued lattice-ordered groups)**.**

```
formulas(assumptions).
% lattice-ordered group
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.
x < y -> ((x * z < y * z) & (z * x < z * y)).

%
((x v x') * (y v y')) < (((y v y') * (y v y')) * ((x v x') * (x v x'))).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/72

Teoria 8.2.72 (#73 Ockham algebras).

```
formulas(assumptions).
% bounded distributive lattice
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).

% dual endomorphism
(x ^ y)' = x' v y'.
(x v y)' = x' ^ y'.
0' = 1.
1' = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/73

Teoria 8.2.73 (#74 Order algebras).

```
formulas(assumptions).
x * x = x.
(x * y) * x = y * x.
(x * y) * y = x * y.
x * ((x * y) * z) = x * (y * z).
((x * y) * z) * y = (x * z) * y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/74

Teoria 8.2.74 (#75 Ordered monoids).

```
formulas(assumptions).
% partially ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
% linear
x < y | y < x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/75

Teoria 8.2.75 (#76 Ordered monoids with zero)**.**

```
formulas(assumptions).
% ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
% zero
x * 0 = 0 & 0 * x = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/76

Teoria 8.2.76 (#77 Ordered semigroup)**.**

```
formulas(assumptions).
% partially ordered semigroup
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
% linear
x < y | y < x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/77

Teoria 8.2.77 (#78 Ordered semilattice)**.**

```
formulas(assumptions).
% ordered semigroup
(x * y) * z = x * (y * z).
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
x < y | y < x.
```

```
% commutative
x * y = y * x.
% idempotent
x * x = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/78

Teoria 8.2.78 (#79 Ore domains)**.**

```
formulas(assumptions).
% ring with identity
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + -x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

% integral
(x * y = 0) -> (x = 0 | y = 0).

% nonzero common multiples exist
(x != 0 & y != 0) -> (exists u exists v ((x * u = y * v) & (y * v != 0)) & exists u exists v ((u * x = v * y) & (v * y
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/79

Teoria 8.2.79 (#80 Ortholattices)**.**

```
formulas(assumptions).
% lattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
% bounded
0 v x = x.
1 v x = 1.
% complemented
x v x' = 1.
x ^ x' = 0.
% de morgan
(x v y)' = x' ^ y'.
(x ^ y)' = x' v y'.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/80

Teoria 8.2.80 (#81 Orthomodular lattices)**.**

```
formulas(assumptions).
% lattice
```

```
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
% bounded
0 v x = x.
1 v x = 1.
% complemented
x v x' = 1.
x ^ x' = 0.
% de morgan
(x v y)' = x' ^ y'.
(x ^ y)' = x' v y'.
% identity
x v (x' ^ (x v y)) = x v y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/81

Teoria 8.2.81 (#82 Partially ordered groups)**.**

```
formulas(assumptions).
% group
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x * x' = 1.
x' * x = 1.
% partially ordered set
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
% order-preserving
x < y -> ((x * z < y * z) & (z * x < z * y)).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/82

Teoria 8.2.82 (#83 Partially ordered monoids)**.**

```
formulas(assumptions).
% monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
% partially ordered set
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
% order-preserving
x < y -> ((x * z < y * z) & (z * x < z * y)).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/83

Teoria 8.2.83 (#84 Partially ordered semigroups)**.**

```
formulas(assumptions).
% semigroup
(x * y) * z = x * (y * z).
% partially ordered set
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
% order-preserving
x < y -> ((x * z < y * z) & (z * x < z * y)).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/84

Teoria 8.2.84 (#85 Pocrims)**.**

```
formulas(assumptions).
% BCK algebra
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
x * 0 = x.
%
(x * y) * z = x * (z @ y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/85

Teoria 8.2.85 (#86 Posets)**.**

```
formulas(assumptions).
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/86

Teoria 8.2.86 (#87 Quandles)**.**

```
formulas(assumptions).
% left-selfdistributive
x v (y v z) = (x v y) v (x v z).
% right-selfdistributive
(x ^ y) ^ z = (x ^ z) ^ (y ^ z).
%
(x v y) ^ x = y.
x v (y ^ x) = y.
x v x = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/87

Teoria 8.2.87 (#88 Quasi-MV-algebra)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z).
x'' = x.
x + 1 = 1.
(x' + y)' + y = (y' + x)' + x.
(x + 0)' = x' + 0.
(x + 0) + 0 = x + 0.
0' = 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/88

Teoria 8.2.88 (#89 Quasi-ordered sets)**.**

```
formulas(assumptions).
x < x.
((x < y) & (y < z)) -> (x < z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/89

Teoria 8.2.89 (#90 Residuated lattices)**.**

```
formulas(assumptions).
% monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
% lattice
(x v y) v z = x v (y v z).
(x ^ y) ^ z = x ^ (y ^ z).
x v y = y v x.
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
% \ left residual of *
(y < x \ z) <-> (x * y < z).
% / right residual of *
(x < z / y) <-> (x * y < z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/90

Teoria 8.2.90 (#91 Residuated partially ordered monoids)**.**

```
formulas(assumptions).
% partially ordered set
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.

% monoid
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
```

```
% \ left residual of *
(x * y < z) <-> (y < x \ z).
% / right residual
(x * y < z) <-> (x < z / y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/91

Teoria 8.2.91 (#92 Residuated posets)**.**

```
formulas(assumptions).
% residuated partially ordered monoid
x < x.
x < y & y < z -> x < z.
x < y & y < x -> x = y.
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
(x * y < z) <-> (y < x \ z).
(x * y < z) <-> (x < z / y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/92

Teoria 8.2.92 (#93 Right hoops)**.**

```
formulas(assumptions).
x * y = y * x.
x * 1 = x.
x / (y * z) = (x / z) / y.
x / x = 1.
(x / y) * y = (y / x) * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/93

Teoria 8.2.93 (#94 Semigroups with zero)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x * 0 = 0.
0 * x = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/94

Teoria 8.2.94 (#95 Semirings with identity)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x + y = y + x.
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
```

```
1 * x = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/95

Teoria 8.2.95 (#96 Semirings with identity and zero)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x + y = y + x.
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.
x * 0 = 0.
0 * x = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/96

Teoria 8.2.96 (#97 Semirings with zero)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x + y = y + x.
x * (y + z) = (x * y) + (x *z).
(x + y) * z = (x * z) + (y * z).
x * 0 = 0.
0 * x = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/97

Teoria 8.2.97 (#98 Sequential algebras)**.**

```
formulas(assumptions).
% boolean algebra
(x v y) v z = x v (y v z).
x v y = y v x.
(x ^ y) ^ z = x ^ (y ^ z).
x ^ y = y ^ x.
(x v y) ^ x = x.
(x ^ y) v x = x.
0 v x = x.
1 v x = 1.
x ^ (y v z) = (x ^ y) v (x ^ z).
x ^ -x = 0.
x v -x = 1.

% monoid
(x @ y) @ z = x @ (y @ z).
e @ x = x.
x @ e = x.

% right-conjugate
(x @ y) ^ z = 0 <-> (x \ z) ^ y = 0.

% left-conjugate
```

```
(x @ y) ^ z = 0 <-> (z / y) ^ x = 0.

% balanced
x \ e = e / x.

% euclidean
x * (y / z) < (x * y) / z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/98

Teoria 8.2.98 (#99 Shells)**.**

```
formulas(assumptions).
0 + x = x.
x + 0 = x.
1 * x = x.
x * 1 = x.
0 * x = 0.
x * 0 = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/99

Teoria 8.2.99 (#100 Skew lattices)**.**

```
formulas(assumptions).
% band
(x v y) v z = x v (y v z).
x v x = x.

% band
(x ^ y) ^ z = x ^ (y ^ z).
x ^ x = x.

% absorption
x ^ (x v y) = x & x = x v (x ^ y).
(x v y) ^ y = y & y = (x ^ y) v y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/100

Teoria 8.2.100 (#101 Skew-fields)**.**

```
formulas(assumptions).
% ring with identity
(x + y) + z = x + (y + z).
x + y = y + x.
x + 0 = x.
x + - x = 0.
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
x * 1 = x.
1 * x = x.

% non-trivial
0 != 1.
```

```
% every non-zero element has a multiplicative inverse
x != 0 -> exists y (x * y = 1).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/101

Teoria 8.2.101 (#102 Tarski algebras)**.**

```
formulas(assumptions).
(x @ y) @ x = x.
(x @ y) @ y = (y @ x) @ x.
x @ (y @ z) = y @ (x @ z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/102

Teoria 8.2.102 (#103 Totally ordered monoids)**.**

```
formulas(assumptions).
% partially ordered monoid
(x * y) * z = x * (y * z).
x * 1 = x.
1 * x = x.
x < x.
x < y & y < x -> x = y.
x < y & y < z -> x < z.
x < y -> ((x * z < y * z) & (z * x < z * y)).
% linear
x < y | y < x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/103

Teoria 8.2.103 (#104 Wajsberg hoops)**.**

```
formulas(assumptions).
% hoop
x * y = y * x.
(x * y) * z = x * (y * z).
1 * x = x.
x @ (y @ z) = (x * y) @ z.
x @ x = 1.
(x @ y) * x = (y @ x) * y.
%
(x @ y) @ y = (y @ x) @ x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/104

Teoria 8.2.104 (#105 Near-semiring)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z).
```

```
(x * y) * z = x * (y * z).
x * (y + z) = (x * y) + (x * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/105

Teoria 8.2.105 (#106 MV-algebra).

```
formulas(assumptions).
x + y = y + x.
(x + y) + z = x + (y + z).
x + x' = 1.
x + 0 = x.
(x')' = x.
0' = 1.
x + 1 = 1.
(x' + y)' + y = (x + y')' + x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/106

Teoria 8.2.106 (#107 Orthoimplication algebra).

```
formulas(assumptions).
(x * y) * x = x.
(x * y) * y = (y * x) * x.
x * ((y * x) * z) = x * z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/107

Teoria 8.2.107 (#108 Orthomodular implication algebra).

```
formulas(assumptions).
x * x = 1.
x * (y * x) = 1.
(x * y) * x = x.
(x * y) * y = (y * x) * x.
(((x * y) * y) * z) * (x * z) = 1.
(((((((((x * y) * y) * z) * z) * z) * x) * x) * z) * x) * x = (((x * y) * y) * z) * z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/108

Teoria 8.2.108 (#109 Quasi-implication algebra).

```
formulas(assumptions).
(x * y) * x = x.
(x * y) * (x * z) = (y * x) * (y * z).
((x * y) * (y * x)) * x = ((y * x) * (x * y)) * y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/109

Teoria 8.2.109 (#110 Idempotent semiring)**.**

```
formulas(assumptions).
% commutative monoid with idempotent addition
x + (y + z) = (x + y) + z.
x + y = y + x.
x + 0 = x.
0 + x = x.
x + x = x.
% monoid
x * (y * z) = (x * y) * z.
x * 1 = x.
1 * x = x.
% distributive
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
% 0 annihilator
x * 0 = 0.
0 * x = 0.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/110

Teoria 8.2.110 (#111 Concurrent semiring)**.**

```
formulas(assumptions).
%%%%% Idempotent semiring + *

% commutative monoid with idempotent addition
x + (y + z) = (x + y) + z.
x + y = y + x.
x + 0 = x.
0 + x = x.
x + x = x.
% monoid
x * (y * z) = (x * y) * z.
x * 1 = x.
1 * x = x.
% distributive
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
% 0 annihilator
x * 0 = 0.
0 * x = 0.
% every idempotent semiring is partially ordered by
x < y <-> x + y = y.

%%%%% Idempotent semiring + @

% commutative monoid with idempotent addition
% x + (y + z) = (x + y) + z.
% x + y = y + x.
% x + 0 = x.
% 0 + x = x.
% x + x = x.
% monoid
x @ (y @ z) = (x @ y) @ z.
x @ 1 = x.
1 @ x = x.
% distributive
x @ (y + z) = (x @ y) + (x @ z).
(x + y) @ z = (x @ z) + (y @ z).
% 0 annihilator
x @ 0 = 0.
```

```
0 @ x = 0.

%
(w * x) @ (y * z) < (w @ y) * (x @ z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/111

Teoria 8.2.111 (#112 Kleene algebra).

```
formulas(assumptions).
%%%%% Idempotent semiring

% commutative monoid with idempotent addition
x + (y + z) = (x + y) + z.
x + y = y + x.
x + 0 = x.
0 + x = x.
x + x = x.
% monoid
x * (y * z) = (x * y) * z.
x * 1 = x.
1 * x = x.
% distributive
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
% 0 annihilator
x * 0 = 0.
0 * x = 0.
% every idempotent semiring is partially ordered by
x < y <-> x + y = y.
% star operator s satisfies the unfold and induction laws
(1 + x) * s(x) = s(x).
(1 + s(x)) * x = s(x).
(z + x) * y < y -> s(x) * z < y.
(z + y) * x < y -> z * s(x) < y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/112

Teoria 8.2.112 (#113 Concurrent Kleene algebra).

```
formulas(assumptions).
%%%%% concurrent semiring +, *, ;

% Idempotent semiring + *

% commutative monoid with idempotent addition
x + (y + z) = (x + y) + z.
x + y = y + x.
x + 0 = x.
0 + x = x.
x + x = x.
% monoid
x * (y * z) = (x * y) * z.
x * 1 = x.
1 * x = x.
% distributive
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
% 0 annihilator
x * 0 = 0.
0 * x = 0.
```

```
% every idempotent semiring is partially ordered by
x < y <-> x + y = y.

% Idempotent semiring +, @

% commutative monoid with idempotent addition
% x + (y + z) = (x + y) + z.
% x + y = y + x.
% x + 0 = x.
% 0 + x = x.
% x + x = x.
% monoid
x @ (y @ z) = (x @ y) @ z.
x @ 1 = x.
1 @ x = x.
% distributive
x @ (y + z) = (x @ y) + (x @ z).
(x + y) @ z = (x @ z) + (y @ z).
% 0 annihilator
x @ 0 = 0.
0 @ x = 0.

%
(w * x) @ (y * z) < (w @ y) * (x @ z).

% Kleene algebra x, *, s

% Idempotent semiring

% commutative monoid with idempotent addition
% x + (y + z) = (x + y) + z.
% x + y = y + x.
% x + 0 = x.
% 0 + x = x.
% x + x = x.
% monoid
% x * (y * z) = (x * y) * z.
% x * 1 = x.
% 1 * x = x.
% distributive
% x * (y + z) = (x * y) + (x * z).
% (x + y) * z = (x * z) + (y * z).
% 0 annihilator
% x * 0 = 0.
% 0 * x = 0.
% every idempotent semiring is partially ordered by
% x < y <-> x + y = y.
% star operator s satisfies the unfold and induction laws
(1 + x) * s(x) = s(x).
(1 + s(x)) * x = s(x).
(z + x) * y < y -> s(x) * z < y.
(z + y) * x < y -> z * s(x) < y.

% Kleene algebra +, @, t

% Idempotent semiring

% commutative monoid with idempotent addition
% x + (y + z) = (x + y) + z.
% x + y = y + x.
% x + 0 = x.
% 0 + x = x.
% x + x = x.
% monoid
x @ (y @ z) = (x @ y) @ z.
x @ 1 = x.
1 @ x = x.
% distributive
x @ (y + z) = (x @ y) + (x @ z).
(x + y) @ z = (x @ z) + (y @ z).
% 0 annihilator
x @ 0 = 0.
```

```
0 @ x = 0.
% every idempotent semiring is partially ordered by
% x < y <-> x + y = y.
% star operator t satisfies the unfold and induction laws
(1 + x) @ t(x) = t(x).
(1 + t(x)) @ x = t(x).
(z + x) @ y < y -> t(x) @ z < y.
(z + y) @ x < y -> z @ t(x) < y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/113

Teoria 8.2.113 (#114 Omega algebra)**.**

```
formulas(assumptions).
%%%%% Kleene algebra

% Idempotent semiring

% commutative monoid with idempotent addition
x + (y + z) = (x + y) + z.
x + y = y + x.
x + 0 = x.
0 + x = x.
x + x = x.
% monoid
x * (y * z) = (x * y) * z.
x * 1 = x.
1 * x = x.
% distributive
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
% 0 annihilator
x * 0 = 0.
0 * x = 0.
% every idempotent semiring is partially ordered by
x < y <-> x + y = y.
% star operator s satisfies the unfold and induction laws
(1 + x) * s(x) = s(x).
(1 + s(x)) * x = s(x).
(z + x) * y < y -> s(x) * z < y.
(z + y) * x < y -> z * s(x) < y.

% omega operator w satisfies the unfold and induction laws
w(x) < x * w(x).
y < (z + x) * y -> y < (w(x) + s(x)) * z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/114

Teoria 8.2.114 (#115 Concurrent semiring with invariants)**.**

```
formulas(assumptions).
%%%%% concurrent semiring

%% Idempotent semiring + *

% commutative monoid with idempotent addition
x + (y + z) = (x + y) + z.
x + y = y + x.
x + 0 = x.
0 + x = x.
x + x = x.
```

```
% monoid
x * (y * z) = (x * y) * z.
x * 1 = x.
1 * x = x.
% distributive
x * (y + z) = (x * y) + (x * z).
(x + y) * z = (x * z) + (y * z).
% 0 annihilator
x * 0 = 0.
0 * x = 0.
% every idempotent semiring is partially ordered by
x < y <-> x + y = y.

%% Idempotent semiring + @

% commutative monoid with idempotent addition
% x + (y + z) = (x + y) + z.
% x + y = y + x.
% x + 0 = x.
% 0 + x = x.
% x + x = x.
% monoid
x @ (y @ z) = (x @ y) @ z.
x @ 1 = x.
1 @ x = x.
% distributive
x @ (y + z) = (x @ y) + (x @ z).
(x + y) @ z = (x @ z) + (y @ z).
% 0 annihilator
x @ 0 = 0.
0 @ x = 0.

%
(w * x) @ (y * z) < (w @ y) * (x @ z).

%%%%% i is a closure operator
1 < i(x).
i(x * y) < i(x + y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/115

Teoria 8.2.115 (#116 Ground mereotopologies)**.**

```
formulas(assumptions).
P(x,x).
P(x,y) & P(y,x) -> x = y.
P(x,y) & P(y,z) -> P(x,z).
C(x,x).
C(x,y) -> C(y,x).
C(z,x) & P(x,y) -> C(y,z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/116

Teoria 8.2.116 (#117 Domain semigroup)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
```

```
d(x + y) = d(x) + d(y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/117

Teoria 8.2.117 (#118 Domain monoid)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x = d(x) * x.
d(x * y) = d(x * d(y)).
d(x) + 1 = 1.
d(0) = 0.
d(x + y) = d(x) + d(y).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/118

Teoria 8.2.118 (#119 Left closure semigroup)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/119

Teoria 8.2.119 (#120 Left closure monoid)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
d(x) * x = x.
d(d(x)) = d(x).
d(x) * d(x * y) = d(x * y).
d(x) * d(y) = d(y) * d(x).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/120

Teoria 8.2.120 (#121 Inverse semigroup)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x'' = x.
(x * x') * x = x.
((x * x') * y) * y' = ((y * y') * x) * x'.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/121

Teoria 8.2.121 (#122 Domain range semigroup).

```
formulas(assumptions).
d(x) * x = x.
d(x * y) = d(x * d(y)).
d(d(x) * y) = d(x) * d(y).
d(x) * d(y) = d(y) * d(x).
d(k(x)) = k(x).
x * k(x) = x.
k(x * y) = k(k(x) * y).
k(x * k(y)) = k(x) * k(y).
k(x) * k(y) = k(y) * k(x).
k(d(x)) = d(x).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/122

Teoria 8.2.122 (#123 Antidomain monoid).

```
formulas(assumptions).
(x * y) * z = x * (y * z).
1 * x = x.
x * 1 = x.
x' * x = 0.
x * 0 = 0.
x' * y' = y' * x'.
x'' * x = x.
x' = (x * y)' * (x * y')'.
(x * y)' * x = ((x * y)' * x) * y'.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/123

Teoria 8.2.123 (#124 Semilattice pseudo-complemented semigroup).

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/124

Teoria 8.2.124 (#125 Closable semilattice pseudo-complemented semigroup).

```
formulas(assumptions).
% semilattice pseudo-complemented semigroup
(x * y) * z = x * (y * z).
x' * x = 0.
x * 0' = x.
x' * y' = y' * x'.
x' * (x' * y)' = x' * y'.
```

```
%
x'' * x = x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/125

Teoria 8.2.125 (#126 Cancellative semigroup)**.**

```
formulas(assumptions).
(x * y) * z = x * (y * z).
x * z = y * z -> x = y.
x * y = x * z -> y = z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/126

Teoria 8.2.126 (#127 Involutory quandle)**.**

```
formulas(assumptions).
x * x = x.
(x * y) * y = x.
(x * y) * z = (x * z) * (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/127

Teoria 8.2.127 (#128 Left Bol loop)**.**

```
formulas(assumptions).
% loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
%
x * (y * (x * z)) = (x * (y * x)) * z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/128

Teoria 8.2.128 (#129 Right Bol loop)**.**

```
formulas(assumptions).
% loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
%
z * ((x * y) * x) = ((z * x) * y) * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/129

Teoria 8.2.129 (#130 Left conjugacy closed loop)**.**

```
formulas(assumptions).
% loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
%
z * (y * x) = ((z * y) / z) * (z * x).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/130

Teoria 8.2.130 (#131 Right conjugacy closed loop)**.**

```
formulas(assumptions).
% loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
%
(x * y) * z = (x * z) * (z \ (y * z)).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/131

Teoria 8.2.131 (#132 Conjugacy closed loop)**.**

```
formulas(assumptions).
% loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
%
z * (y * x) = ((z * y) / z) * (z * x).
(x * y) * z = (x * z) * (z \ (y * z)).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/132

Teoria 8.2.132 (#133 RIF loop)**.**

```
formulas(assumptions).
% loop
x \ (x * y) = y.
```

```
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
%
(x * y) * (z * (x * y)) = ((x * (y * z)) * x) * y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/133

Teoria 8.2.133 (#134 ARIF loop)**.**

```
formulas(assumptions).
% loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
%
(z * x) * ((y * x) * y) = (z * ((x * y) * x)) * y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/134

Teoria 8.2.134 (#135 Extra loop)**.**

```
formulas(assumptions).
% loop
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
x * 1 = x.
x = 1 * x.
%
(x * (y * z)) * y = (x * y) * (z * y).
% (y * z) * (y * x) = y * ((z * y) * x).
% ((x * y) * z) * x = x * (y * (z * x)).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/135

Teoria 8.2.135 (#136 F-quasigroup)**.**

```
formulas(assumptions).
% quasigroup
x \ (x * y) = y.
x * (x \ y) = y.
(x * y) / y = x.
(x / y) * y = x.
% left F-law
x * (y * z) = (x * y) * ((x \ x) * z).
% right F-law
(z * y) * x = (z * (x / x)) * (y * x).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/136

Teoria 8.2.136 (#137 Steiner quasigroup).

```
formulas(assumptions).
x * x = x.
x * y = y * x.
x * (x * y) = y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/137

Teoria 8.2.137 (#138 Steiner loop).

```
formulas(assumptions).
x * x = 1.
x * y = y * x.
x * (x * y) = y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/138

Teoria 8.2.138 (#139 Rectangular quasigroup).

```
formulas(assumptions).
x \ (x * x) = x.
(x * x) / x = x.
x * (x \ y) = x \ (x * y).
(x / y) * y = (x * y) / y.
(x \ y) \ ((x \ y) * (z * u)) = (x \ (x * z)) * u.
((x * y) * (z / u)) / (z / u) = x * ((y * u) / u).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/139

Teoria 8.2.139 (#140 Rectangular loop).

```
formulas(assumptions).
% rectangular quasigroup
x \ (x * x) = x.
(x * x) / x = x.
x * (x \ y) = x \ (x * y).
(x / y) * y = (x * y) / y.
(x \ y) \ ((x \ y) * (z * u)) = (x \ (x * z)) * u.
((x * y) * (z / u)) / (z / u) = x * ((y * u) / u).
%
x \ (x * (y \ y)) = ((x / x) * y) / y.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/140

Teoria 8.2.140 (#141 Trisemigroup).

```
formulas(assumptions).
x * (y * z) = (x + y) * z.
x * (y * z) = (x @ y) * z.
x * (y + z) = (x * y) + z.
x * (y @ z) = (x * y) @ z.
x + (y + z) = x + (y * z).
x + (y + z) = x + (y @ z).
(x @ y) + z = x @ (y + z).
(x + y) @ z = x @ (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/141

Teoria 8.2.141 (#142 Trigroup).

```
formulas(assumptions).
% trisemigroup
x * (y * z) = (x + y) * z.
x * (y * z) = (x @ y) * z.
x * (y + z) = (x * y) + z.
x * (y @ z) = (x * y) @ z.
x + (y + z) = x + (y * z).
x + (y + z) = x + (y @ z).
(x @ y) + z = x @ (y + z).
(x + y) @ z = x @ (y * z).
%
1 * x = x.
x = x + 1.
x * x' = 1.
1 = x' + x.
x @ x' = 1.
1 = x' @ x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/142

Teoria 8.2.142 (#144 Double Ward quasigroup).

```
formulas(assumptions).
e * e = e.
(e * (x * z)) * ((e * y) * z) = x * y.
(y * x) * y = x & y * (x * y) = x.
e * x = x * e.
x * (x * e) = e & (e * x) * x = e.
x * y = e * ((e * y) * (e * x)).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/144

Teoria 8.2.143 (#145 Digroup).

```
formulas(assumptions).
(x * y) * z = x * (y * z).
(x + y) + z = x + (y + z).
x * (y + z) = x * (y * z).
(x * y) + z = (x + y) + z.
x + (y * z) = (x + y) * z.
1 + x = x & x = x * 1.
```

```
x + x' = 1 & x' * x = 1.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/145

Teoria 8.2.144 (#146 Left disemigroup).

```
formulas(assumptions).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/146

Teoria 8.2.145 (#147 Right disemigroup).

```
formulas(assumptions).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/147

Teoria 8.2.146 (#148 Disemigroup).

```
formulas(assumptions).
% Left disemigroup
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
x + (y + z) = (x * y) + z.
x + (y * z) = (x + y) * z.
% Right disemigroup
% (x + y) + z = x + (y + z).
% (x * y) * z = x * (y * z).
x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/148

Teoria 8.2.147 (#149 Left I-Semiring).

```
formulas(assumptions).
%commutative additive monoid
x + y = y + x.
x + 0 = x.
x + (y + z) = (x + y) + z.
%multiplicative monoid
x @ 1 = x & 1 @ x = x.
x @ (y @ z) = (x @ y) @ z.
```

```
%left annihilation
0 @ x = 0.
%idempotence
x + x = x.
%left distributivity
(x + y) @ z = (x @ z) + (y @ z).
%subsumption order
x < y <-> x + y = y.
%right superdistributivity
(x @ y) + (x @ z) < x @ (y + z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/149

Teoria 8.2.148 (#150 Doppelsemigroup)**.**

```
formulas(assumptions).
(x + y) * z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/150

Teoria 8.2.149 (#151 Strong doppelsemigroup)**.**

```
formulas(assumptions).
(x + y) * z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).

x + (y * z) = x * (y + z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/151

Teoria 8.2.150 (#152 Dimonoid)**.**

```
formulas(assumptions).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/152

Teoria 8.2.151 (#153 Commutative dimonoid)**.**

```
formulas(assumptions).
(x * y) + z = x * (y + z).
(x + y) + z = x + (y * z).
(x + y) * z = x * (y * z).
```

```
x + y = y + x.
x * y = y * x.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/153

Teoria 8.2.152 (#154 Diassociative semigroup)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z) & x + (y + z) = x + (y * z).
(x * y) + z = x * (y + z).
(x * y) * z = (x + y) * z & (x + y) * z = x * (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/154

Teoria 8.2.153 (#155 Duplicial semigroup)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z).
(x * y) + z = x * (y + z).
(x * y) * z = x * (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/155

Teoria 8.2.154 (#156 Extended duplicial semigroup)**.**

```
formulas(assumptions).
% Duplicial semigroup
(x + y) + z = x + (y + z).
(x * y) + z = x * (y + z).
(x * y) * z = x * (y * z).
%
x v (y + z) = x v y.
(x * y) ^ z = y ^ z.
(x ^ y) + ((x + y) ^ z) = x ^ (y + z).
(x ^ y) ^ ((x + y) ^ z) = y ^ z.
(x v (y * z)) v (y v z) = x v y.
(x v (y * z)) * (y v z) = (x * y) v z.
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/156

Teoria 8.2.155 (#157 Associative dialgebra)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z).
(x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z).
(x * y) * z = x * (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/157

Teoria 8.2.156 (#158 Associative trialgebra)**.**

```
formulas(assumptions).
(x + y) + z = x + (y + z).
(x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z).
(x * y) * z = x * (y * z).

(x + y) + z = x + (y @ z).
(x @ y) + z = x @ (y + z).
(x + y) @ z = x @ (y * z).
(x * y) @ z = x * (y @ z).
(x @ y) * z = x * (y * z).

(x @ y) @ z = x @ (y @ z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/158

Teoria 8.2.157 (#159 Trialgebra)**.**

```
formulas(assumptions).
(x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z).
(x + y) + z = x + (y @ z).
(x @ y) + z = x @ (y + z).
(x + y) @ z = x @ (y * z).
(x * y) @ z = x * (y @ z).
(x @ y) * z = x * (y * z).
end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/159

Teoria 8.2.158 (#160 Associative dimonoid)**.**

```
formulas(assumptions).
x + (y + z) = (x + y) + z & (x + y) + z = x + (y * z).
(x * y) + z = x * (y + z).
(x + y) * z = x * (y * z) & x * (y * z) = (x * y) * z.

end_of_list.
formulas(goals).
end_of_list.
```

https://marciedb.pythonanywhere.com/definition/160

8.3 Package CREAM

arXiv:2202.00613v1 [math.GR] 1 Feb 2022

CREAM: A PACKAGE TO COMPUTE [AUTO, ENDO, ISO, MONO, EPI]-MORPHISMS, CONGRUENCES, DIVISORS AND MORE FOR ALGEBRAS OF TYPE $(2^n, 1^n)$

JOÃO ARAÚJO, RUI BARRADAS PEREIRA, WOLFRAM BENTZ, CHOIWAH CHOW, JOÃO RAMIRES, LUIS SEQUEIRA, AND CARLOS SOUSA

ABSTRACT. The CREAM GAP package computes automorphisms, congruences, endomorphisms and subalgebras of algebras with an arbitrary number of binary and unary operations; it also decides if between two such algebras there exists a monomorphism, an epimorphism, an isomorphism or if one is a divisor of the other. Thus it finds those objects for almost all algebras used in practice (groups, quasigroups in their various signatures, semigroups possibly with many unary operations, fields, semi-rings, quandles, logic algebras, etc).

As a one-size-fits-all package, it only relies on universal algebra theorems, without taking advantage of specific theorems about, eg, groups or semigroups to reduce the search space. Canon and Holt produced very fast code to compute automorphisms of groups that outperform CREAM on orders larger than 128. Similarly, Mitchell et al. take advantage of deep theorems to compute automorphisms and congruences of completely 0-simple semigroups in a very efficient manner. However these domains (groups of order above 128 and completely 0-simple semigroups) are among the very few examples of GAP code faster than our general purpose package CREAM. For the overwhelming majority of other classes of algebras, either ours is the first code computing the above mentioned objects, or the existing algorithms are outperformed by CREAM, in some cases by several orders of magnitude.

To get this performance, CREAM uses a mixture of universal algebra algorithms together with GAP coupled with artificial intelligence theorem proving tools (AITP) and very delicate C implementations. As an example of the latter, we re-implement Freese's very clever algorithm for computing congruences in universal algebras, in a way that outperforms all other known implementations.

1. INTRODUCTION

Investigation of automorphism groups of mathematical structures is one of the classical algebraic problems. A cornerstone was the work of Evariste Galois, but its impact goes far beyond. In the words of P. J. Cameron [22]:

> *In the famous Erlanger Programme, Klein proposed that geometry is the study of symmetry; more precisely, the geometric properties of an object are those which are invariant under all automorphisms of the objects. (. . .) According to Artin, "the investigation of symmetries of a given mathematical structure has always yielded the most powerful results."*

These ideas, expressed by Klein in 1872 [36] and by Artin in 1957 [18], give some insight on why the computation of automorphisms of various mathematical structures has been such an attractive topic for so many

MATHEMATICS DEPARTMENT, FACULTY OF SCIENCE AND TECHNOLOGY, UNIVERSIDADE NOVA DE LISBOA, PORTUGAL
DEPARTMENT OF SCIENCE AND TECHNOLOGY, UNIVERSIDADE ABERTA, LISBON, PORTUGAL
DEPARTMENT OF SCIENCE AND TECHNOLOGY, UNIVERSIDADE ABERTA, LISBON, PORTUGAL
DEPARTMENT OF SCIENCE AND TECHNOLOGY, UNIVERSIDADE ABERTA, LISBON, PORTUGAL
DEPARTMENT OF SCIENCE AND TECHNOLOGY, UNIVERSIDADE ABERTA, LISBON, PORTUGAL
MATHEMATICS DEPARTMENT, UNIVERSIDADE DE LISBOA, LISBON, PORTUGAL
DEPARTMENT OF SCIENCE AND TECHNOLOGY, UNIVERSIDADE ABERTA, LISBON, PORTUGAL
E-mail addresses: jj.araujo@fct.unl.pt, rmbper@gmail.com, wolfram.bentz@uab.pt, choiwah.chow@gmail.com, joao.j.ramires@gmail.com, lfsequeira@fc.ul.pt, cfmsousa@sapo.pt.

decades. It should be observed here that this effort is closely linked to an omnipresent problem in mathematics – stated explicitly by Ulam [60] – of describing the mathematical objects whose endomorphisms encode *all the relevant information* about them:

$$\mathrm{End}(A) \cong \mathrm{End}(B) \Rightarrow A \cong B.$$

The connection between endomorphisms and congruences is well known, but the importance of congruences goes far beyond a tool to get endomorphisms; in fact their importance in modern algebra can hardly be overestimated.

Congruences have close ties to the structure of algebras - for example, the subdirectly irreducible algebras are precisely the ones that possess a unique minimal nontrivial congruence; direct irreducibility is also characterized by properties of congruences. In Universal Algebra, a variety is a class of algebras of the same type, that satisfy a certain set of identities, or equivalently, by a celebrated theorem of Garrett Birkhoff (see [21, Theorem II.11.9]), a class closed under taking subalgebras, homomorphic images and direct products. Each variety is also generated by its subdirectly irreducible algebras. Properties of congruence lattices of algebras over a variety are closely related to the identities satisfied in that variety. This gave rise to what is usually referred to the theory of Maltsev conditions - see, for example, [31]. The commutator of normal subgroups was generalized for algebras, first those in a congruence-permutable variety [55], then for any algebra in a congruence modular variety [33]; see [29]. Thus notions like Abelianness or nilpotency can be applied meaningfully in a much wider context, and many fruitful consequences have been derived, for example in the context of natural dualities [24]. Tame Congruence Theory [34] emerged as a powerful tool to study the structure of finite algebras, and continues to provide a big leverage in the study of locally finite varieties. These examples point to the critical importance of the study of congruences in algebra.

As observed above, congruences are closely related to endomorphisms and the latter form a monoid (semigroup with identity) that very often encodes very relevant information about the original object and can be investigated with the increasingly powerful techniques developed by experts in semigroup theory (especially since they started to massively apply the classification of finite simple groups [1, 3–8, 10–12, 19, 49, 51, 54, 57]).

Everything said above should make the case in favour of a general centralized and very effective GAP tool that computes congruences, automorphisms and endomorphisms of general algebras of type $(2^m, 1^n)$. We chose these since the overwhelming majority of algebras used in practice have this type: it is a very rare occurrence that an algebraic structure actually uses ternary or higher arity operations (see [21]).

The problem is that computing [endo]automorphisms/congruences of general algebras is a difficult task and hence the majority of existing tools target specific classes (groups, semigroups, quasigroups, etc.) in order to take advantage of the domain's theorems. Probably the best example is the GAP code to find automorphisms of groups, devised by J. J. Cannon and D. Holt [23], that takes advantage of many deep theorems and builds upon many past computational optimizations to quickly get auxiliary objects. In contrast, since we want our algorithms to apply for general algebras of type $(2^m, 1^n)$, we cannot rely on theorems that only hold in very specific cases. The challenge of this project is thus to provide an effective tool to compute the objects for finite algebras of type $(2^m, 1^n)$ that works for all algebras of this type and does not fall far behind the more dedicated tools that take advantage of very specific domain theorems.

The final result is that CREAM compares favourably even with the specialized Cannon and Holt tool referred to above: for groups up to order 128, our general purpose tool finds the automorphisms of a group faster than their very optimized code. For most other classes of algebras, our tool outperforms specialized code, in some cases by large margins. This was achieved by a combination of results and ideas from different parts of computational algebra, artificial intelligence theorem proving, and optimized C implementations. Regarding congruences, we take advantage of a new implementation of Freese's algorithm [28], an algorithm that stands out by its quality and generality; for automorphisms, after many different tests and approaches, the most effective way turned out to be a generalization of the ideas developed in [48]. Once possessing a very good tool to compute automorphisms, finding [mono,epi,iso]-morphisms is straightforward.

Finally, our tool also handles divisors. Given two algebras, A and B, we say that B divides A if B is a homomorphic image of a subalgebra of A. In different words, B divides A if there is a subalgebra C of A and a congruence ρ of C such that C/ρ is isomorphic to B. Like congruences, divisors are very important tools. For example, the celebrated Krohn–Rhodes theorem states that every finite semigroup S divides a wreath product of finite simple groups, each dividing S, together with copies of the 3-element monoid $\{1, a, b\}$, where $ba = aa = a$, $ab = bb = b$ (for details see [53]). The key difficulty when finding divisors is to find the subalgebras of a given algebra; CREAM has an algorithm to compute them.

There are numerous papers concerning the automorphism groups of particular classes of algebras, for example, Schreier [52] and Mal'cev [42] described all automorphisms of the semigroup of all mappings from a set to itself. Similar results have been obtained for various other structures such as orders, equivalence relations, graphs, and hypergraphs; see the survey papers [46] and [47]. More examples are provided, among others, by Gluskĭn [32], Araújo and Konieczny [14], [15], and [16], Fitzpatrick and Symons [26], Levi [38] and [39], Liber [40], Magill [41], Schein [50], Sullivan [58], and Šutov [59].

Fast algorithms exist to find the congruences and automorphisms of groups and semigroups. For example, with GAP [30], the package SEMIGROUPS includes the function CongruencesOfSemigroup that returns the congruences of a given semigroup. This function only works for a limited set of semigroups, belonging to the classes Simple, Brandt, Group, Zero Simple or Rectangular Band, and only for Rees Matrix semigroups and Rees Zero Matrix semigroups is it highly efficient. On a different platform, UACalc [27] supports a wider range of algebras. UACalc also implements the Freeze algorithm that is the basis for the calculation of congruences described here.

Regarding automorphisms in GAP, many special properties of groups are used to implement the function *AutomorphismGroup* to efficiently find the automorphism group of a given group. However, this specialized method, while extremely efficient, works only on groups and not on any other more general algebraic structures, such as quasigroups, semigroups and magmas. Likewise, the Loops package [48] in GAP provides another version of the function *AutomorphismGroup* to compute all the automorphisms of a given quasigroup.

To date, there are no known implementations of general functions for finding divisors, congruences and [endo, auto, epi, mono, iso]-morphisms of magmas or algebras of type $(2^m, 1^n)$ in GAP. To fill the void, the CREAM package implements such functions in GAP with part of the code written in C for performance.

This article is composed of 6 sections: this introduction, a short description of the mathematical concepts and background, the description of the algorithms used in the CREAM package, a comparative discussion of the performance of the package, a list of applications of the package and finally a short conclusion.

2. Mathematical Background

2.1. **Algebra of Type (2^m,1^n).** An Algebra in the sense of Universal Algebra is an algebraic structure consisting of a set A together with a collection of operations on A. (If we want to be more precise, an algebra also contains an indexed scheme that references the operations, but we are not going to enter those technical details; for a complete definition see [21].)

An n-ary operation on A is a function that takes an n-tuple of elements from A and returns a single element of A, that is, a function from A^n to A. The number n is called the *arity* of the operation. For the scope of this package we are only considering operations with arity 1 or 2, i.e., unary and binary operations. An algebra of type (2^m,1^n) is a universal algebra with m binary and n unary operations.

The package represents a finite Universal Algebra as a list of operations. An operation of arity 1 is represented by a vector, while an operation of arity 2 is represented by a square matrix. The underlying set A is implicitly specified by the vector or matrix sizes, which need to agree for a valid representation. If this size is d, the algebra is defined on the set $A = \{1, \ldots, d\}$. We can safely ignore the ambiguity this introduces in the (uninteresting) case of an algebra without operations. Each vector or matrix describes the corresponding operation by listing all images in the obvious way. The following is an example of a representation for an algebra with a unary and a binary operation.

```
[ [3, 1, 2], [ [1, 2, 3], [2, 3, 1], [3, 1, 2] ] ]
```

For those CREAM functions that involve more than one algebra, the operations of the algebras need to be aligned by a common index scheme. In our representation, this index is indirectly provided by the order in which the operations are listed. The involved algebras need to be compatible, that is, they have the same number of operations of each arity, and the operations are listed in the same position in the algebra representation.

2.2. **Partition.** A partition of a set A is a collection of non-empty subsets of A, such that every element of A is included in exactly one subset [21]. Each subset in a partition is called a block or part. For example, $\{\{1,3\},\{2,4\}\}$ is a 2-blocks partition of the set $\{1,2,3,4\}$.

A block of a partition is efficiently represented as a tree. A partition is represented by a forest, that is, a disjoint union of trees. The forest is physically presented in the computer memory by an array, whose size will be the size of the set A.

In a tree, each node has a parent node, except for the top node or root. In the array representation, each position of the array represents a node and the value of the node points to its parent node. Top nodes should have a value indicating that the node is the root of the tree. A negative number is used to signal the root, and the absolute value of this number is the number of elements of the block.

A shallow tree is a tree in which all non-root nodes are connected directly to the root. To obtain a unique representation, we adopt the convention that the trees used in presenting a partition are shallow, and that the smallest value in a block will be the root node. A partition representation that respects the above conventions will be called a normalized (representation of a) partition.

Thus, for $l, m \geq 1$, the array

$$\begin{array}{ccccccc} [& \ldots & , -l, & \ldots & , m, & \ldots &] \\ & & \uparrow & & \uparrow & & \\ & & \text{position } i & & \text{position } k & & \end{array}$$

means that i is the root of a block (as its entry is negative) of size l; in addition the element k belongs to the block whose root is m.

Using these conventions, the encoding of the partition `[[1], [2], [3], [4], [5], [6]]` induced by the identity relation is

```
[-1, -1, -1, -1, -1, -1]
```

while the partition `[[1, 2, 3, 4, 5, 6]]` with just one block is encoded as

```
[-6, 1, 1, 1, 1, 1]
```

Other examples are given below.

partition	encoded as
`[[1, 6], [2], [3, 5], [4]]`	`[-2, -1, -2, -1, 3, 1]`
`[[1, 3, 5], [2, 6], [4]]`	`[-3, -2, 1, -1, 1, 2]`
`[[1, 2, 5, 6], [3, 4]]`	`[-4, 1, -2, 3, 1, 1]`

This non-intuitive way of representing partitions will prove its usefulness later on.

2.3. **Congruences.** A congruence of an algebra is an equivalence relation on its underlying set that is compatible with all algebraic operations [21].

Technically, a congruence relation is an equivalence relation $\equiv$ on an algebra that satisfies $\mu(a_1, a_2, ..., a_n) \equiv \mu(a'_1, a'_2, ..., a'_n)$ for every n-ary operation μ and all elements $a_1, ..., a_n, a'_1, ..., a'_n$ such that $a_i \equiv a'_i$ for each $i = 1, ..., n$.

Congruences will be represented in the same way as their corresponding partitions, as detailed in the previous section.

2.4. **Homomorphism, Endomorphism, Isomorphism and Automorphism.** If A, B are two algebras of the same type, then a function $f : A \to B$ is a *homomorphism* from A to B if $\mu(f(a_1), f(a_2), ..., f(a_n)) = f(\mu(a_1, a_2, ..., a_n))$ for every n-ary operation μ and $a_1, \ldots a_n \in A$ (more precisely, μ here stands for the two operations of A and B that are indexed equally by the (common) index scheme).

An *endomorphism* is a *homomorphism* from an algebra to itself, a *monomorphism* is an injective *homomorphism*, while an *isomorphism* is a bijective *homomorphism*. An *automorphism* is a *homomorphism* that is both an *isomorphism* and an *endomorphism*.

3. The Algorithms

3.1. **Congruences algorithm.** The starting point for the computation of congruences are the algorithms described in Freese [28] to calculate the smallest congruence containing a given partition Θ of a finite algebra A; in particular this is used to compute the smallest congruence containing a pair of elements $(a, b) \in A \times A$, called the *principal congruence* generated by $\{a, b\}$. From this base algorithm all congruences of the algebra A are generated in an efficient way. Optimizations to the original algorithm were introduced taking into account that we only allow operations of arity at most 2. In addition, we take advantage of C to get a faster implementation than the original implementations made by Freese and his collaborators. Finally, our implementation is integrated with GAP and hence fully compatible with its other resources.

3.1.1. *Partition Functions.* There are several partition functions that play an important role in the algorithm to compute principal congruences.

The function **CreamRootBlock** is an operation that returns the root node for a node i. The root node of a node is itself if the value of the node representation is negative. If the value of the node representation points to a different node then that node is the parent node. This algorithm could be run recursively until reaching the root node but is implemented iteratively for better performance. This operation will work both for normalized and non-normalized partitions.

Before returning, the node parent of i is set to the found root node in order to make the tree representing the partition as shallow as possible, thus avoiding that in future calls the algorithm needs to transverse several nodes to reach the root node.

Algorithm 1 CreamRootBlock (i, partition)

$j \leftarrow i$
while $partition[j] \geq 0$ **do**
  $j \leftarrow partition[j]$
end while
if $i \neq j$ **then**
  $partition[i] \leftarrow j$
end if
return j

The function **CreamJoinBlocks** is an operation that joins the blocks containing given elements x and y. This operation will work both for normalized and non-normalized partitions. The resulting partition may not in general be normalized even if the original partition is.

In order to keep the tree representing the partition as shallow as possible, the root node of the merged block will be the root node of the larger original block.

Algorithm 2 CreamJoinBlocks (x, y, partition)

```
r ← CreamRootBlock(x, partition)
s ← CreamRootBlock(y, partition)
if r ≠ s then
  if partition[r] < partition[s] then
    partition[r] ← partition[r] + partition[s]
    partition[s] ← r
  else
    partition[s] ← partition[r] + partition[s]
    partition[r] ← s
  end if
end if
```

The **CreamNumberOfBlocks** function returns the number of blocks of a partition. Given the encoding of partitions we simply count the number of positions of the array that have negative values.

Algorithm 3 CreamNumberOfBlocks (partition)

```
nblocks ← 0
dimension ← ArraySize(partition)
for i = 1 to dimension do
  if partition[i] < 0 then
    nblocks ← nblocks + 1
  end if
end for
return nblocks
```

The **CreamNormalizePartition** function normalizes the partition by making it shallow and having the smallest element of each block the root node.

Algorithm 4 CreamNormalizePartition (partition)

```
dimension ← ArraySize(partition)
for i = 1 to dimension do
  r ← CreamRootBlock(i, partition)
  if r ≥ i then
    partition[i] ← −1
    if r > i then
      partition[r] ← i
    end if
  else
    partition[r] ← partition[r] − 1
  end if
end for
```

The **CreamJoinPartition** function computes the join of two partitions, i.e. the smallest partition containing both input partitions.

Algorithm 5 CreamJoinPartition (partition1, partition2)

$dimension \leftarrow ArraySize(partition1)$
for $i = 1$ to $dimension$ **do**
  CreamJoinBlocks(i,CreamRootBlock (i,$partition1$),$partition2$)
end for

The **CreamComparePartitions** function compares normalized partitions. Returns 0 if the partitions are equal, -1 if partition1 $>$ partition2, and 1 if partition1 $<$ partition2. Partitions are ordered by the order of the underlying list. The purpose of this function is to allow binary search in sets of partitions.

Algorithm 6 CreamComparePartitions (partition1, partition2)

$dimension \leftarrow Length(partition1)$
for $i = 1$ to $dimension - 1$ **do**
  if $partition1[i] > partition2[i]$ **then**
    return -1
  end if
  if $partition1[i] < partition2[i]$ **then**
    return 1
  end if
end for
return 0

3.1.2. *Computing Principal Congruences.* The base algorithm in [28] computes the smallest congruence containing Θ, a partition of a finite algebra A. The simplest case is when Θ only contains one nontrivial block and this block has only two elements.

The algorithm only works with unary operations. Since our algebras have binary operations, we have to convert them to a family of unary operations. In this case a binary operation $f(x, y)$ can be converted into $2n$ unary operations (where n is the size of the algebra) by assigning to each element $x \in A$ the unary operations c_x and r_x, which are induced by the corresponding column and row in the Cayley table of f:

$$c_x(y) = f(y, x) \text{ and } r_x(y) = f(x, y).$$

For example, the algebra:

$$[[[2, 1, 1], [1, 2, 2], [1, 3, 2]]]$$

induces the following unary operations (after removing duplicates):

$$[[2, 1, 1], [1, 2, 2], [1, 3, 2], [1, 2, 3]].$$

The principal congruence algorithm takes as input the algebra and the generating pair of elements.

The algorithm joins the pair of elements in the same block and applies the algebra's unary functions to each element of the pair. In this way, it obtains one additional pair per function that will be joined in the same block, repeating this process until the list of pairs is exhausted.

Algorithm 7 CreamPrincipalCongruence (algebra,InitialPair)

$PairList \leftarrow [InitialPair]$
$partition \leftarrow SingletonPartition()$
$partition \leftarrow CreamJoinBlocks(partition, InitialPair[1], InitialPair[2])$
$NFuncs \leftarrow ArraySize(algebra)$
while $PairList \neq empty$ **do**
  $Pair \leftarrow Pop(PairList)$
  for $i = 1$ to $NFuncs$ **do**
    $f \leftarrow algebra(i)$
    $r \leftarrow CreamRootBlock(partition, f(Pair[1]))$
    $s \leftarrow CreamRootBlock(partition, f(Pair[2]))$
    if $r \neq s$ **then**
      $partition \leftarrow CreamJoinBlocks(partition, r, s)$
      $PairList \leftarrow Push(PairList, [r, s])$
    end if
  end for
end while
return $partition$

As described in [28] this algorithm is very efficient showing a moderate growth of execution time with n. Apart from the algorithm itself, several implementation choices were used to increase efficiency as well.

One of these aspects is the use of arrays to encode partitions which allow for a constant and very fast random access to the partition element.

This is combined with the balancing and collapsing of the trees representing the blocks in the partition that are parts of the **CreamJoinBlocks** and **CreamRootBlock** algorithms. This approach aims for trees that are as shallow as possible during the execution of the **CreamPrincipalCongruence** algorithm.

In **CreamJoinBlocks** the tree is balanced by keeping as root of the joined block the root of the bigger original block.

In **CreamRootBlock** the tree is collapsed by setting the parent node of the element on which the function is called equal to its return value, avoiding having to traverse several nodes of the tree in future calls.

Both of these implementation details allow for shallower trees representing the blocks, which will make **CreamRootBlock** faster, given that fewer nodes will have to be transversed to determine the root node of a node.

This is especially important since **CreamRootBlock** is the most called function when calculating a principal congruence of an algebra.

To compute all the principal congruences this function is called for all pairs of elements of A.

Algorithm 8 CreamAllPrincipalCongruences (algebra)

$dimension \leftarrow SizeAlgebra(algebra)$
$allPrincipalCongruences \leftarrow []$
for $i = 1$ to $dimension - 1$ **do**
  for $j = i + 1$ to $dimension$ **do**
    $congruence \leftarrow CreamPrincipalCongruence(algebra, [i, j])$
    $AddCongruence(allPrincipalCongruences, congruence)$
  end for
end for
return $allPrincipalCongruences$

3.1.3. *Calculating All Congruences.* It is known that the congruences of an algebra form a lattice with the usual set inclusion partial order. The meet is the usual intersection; the join of two congruences is the smallest congruence containing both.

The minimal elements of this lattice are precisely the principal congruences, and we can obtain all congruences by computing joins, starting with the principal congruences.

The key part is to find an efficient way to combine minimal congruences to get all congruences of the algebra. The principal congruences are stored in an ordered set and the congruences are combined from start to end by the join operation. Each congruence resulting from these joins will be added to the ordered set (eliminating duplicates).

Algorithm 9 CreamAllCongruences (algebra)

```
allPrincipalCongruences ← CreamAllPrincipalCongruences(algebra)
allCongruences ← allPrincipalCongruences
i ← 1
while i < Size(allCongruences) do
  for j = 1 to Size(allPrincipalCongruences) do
    if ¬isContained(allPrincipalCongruences[j], allCongruences[i]) then
      congruence ← JoinPartition(allCongruences[i], allPrincipalCongruences[j])
      AddCongruence(allCongruences, congruence)
    end if
  end for
  i ← i + 1
end while
return allCongruences
```

The join operation is only called if the principal congruence is not contained in the congruence that is being joined with. This is optimized by preserving the information about which principal partitions were joined.

3.1.4. *Calculating Monolithic Algebras.* Monolithic Algebras are algebras that have a single minimal congruence that is contained in every other congruence except for the identity congruence.

This is simple to calculate having the list of minimal congruences and a function that compares two partitions and calculates whether one is contained in the other.

The **CreamContainedPartition** function returns whether a partition is contained in another. This function is used internally to determine whether an Algebra is monolithic or not. This algorithm assumes that the partitions are normalized.

Algorithm 10 CreamContainedPartition (partition1, partition2)

```
dimension = ArraySize(partition1)
for i = 1 to dimension do
  if partition1[i] < 0 then
    if partition2[i] < 0 then
      block2 ← i
    else
      block2 ← partition2[i]
    end if
    for j = i + 1 to dimension do
      if partition1[j] = i and not block2 = partition2[j] then
        return false
      end if
    end for
  end if
end for
return true
```

We calculate the principal congruences and compare them eliminating those that contain other congruences. If a single congruence is returned then this congruence is contained in every other congruence (and the algebra is monolithic). This is done with the **CreamIsAlgebraMonolithic** function.

Algorithm 11 CreamIsAlgebraMonolithic (algebra)

```
partitions ← CreamAllPrincipalCongruences(algebra)
npartitions ← Size(partitions)
if npartitions > 1 then
  i ← 2
  repeat
    if CreamContainedPartition(partitions[1], partitions[i]) then
      Remove(partitions, i)
    else if CreamContainedPartition(partitions[i], partitions[1]) then
      partitions[1] ← partitions[i]
      Remove(partitions, i)
      i ← 2
    else
      i ← i + 1
    end if
  until Size (partitions) < i
end if
npartitions ← Size(partitions)
if npartitions > 1 then
  return false
else
  return true
end if
```

3.2. **Automorphism algorithm.** The automorphisms of an algebra A of type (2^m,1^n) can be derived from the calculation of the automorphisms of a set of algebras each containing only one binary operation of

A. The automorphism group of the algebra will be the intersection of all the automorphism groups of these magmas that also commute, function composition-wise, with all the unary operations of the original algebra.

In general, it is difficult to obtain automorphisms efficiently. The authors of the Loops GAP package [48] used an idea that we could apply to general magmas. With some adaptations this allows for very effective computing of the automorphisms of magmas.

3.2.1. *Invariant Vector.* The general idea is to pick a list of properties invariant under homomorphisms and then partition the algebras using these properties. For example, if e is idempotent, then any endomorphism must map e onto an idempotent. This is the basis of the *Discriminator* in the Loops package, which implemented nine such invariants for the domain elements of quasigroups. However, most of these invariants cannot be carried over directly to magmas. We devised a total of seventeen invariants that hold for any magma. In general, unless otherwise said, given an element x in a Magma M and $k \in \mathbb{Z}^+$, we define x^k to be $x^{k-1} * x$ for $k \geq 2$ and $x^1 = x$ (ie. we associate on the left). For each element p in a magma M, CREAM computes the following:

(1) Smallest k such that $p^k = p^n$, with $n > k > 1$.
(2) Number of domain elements for which p is a left identity.
(3) Number of domain elements for which p is a right identity.
(4) Number of elements y such that $p = (py)p$.
(5) Number of distinct elements in row p of the multiplication table.
(6) Number of distinct elements in column p of the multiplication table.
(7) 1 if p is an idempotent, 0 otherwise.
(8) Number of idempotents in column p of the multiplication table.
(9) Number of idempotents in row p of the multiplication table.
(10) 1 if the equality $p(pp) = (pp)p$ holds, 0 otherwise.
(11) Number of domain elements that commute with p.
(12) Number of domain elements s for which $(ss)p = p(ss)$.
(13) Number of domain elements s such that $s^2 = p$.
(14) Number of domain elements s satisfying $p(ps) = (pp)s$
(15) Number of multisets $\{x, y\}$ with $xy = yx = p$.
(16) Number of elements $t \in M$ such that for two idempotents $e, f \in M$, we have $p = et = tf$.
(17) Number of elements $t \in M$ for which there exist two elements $x, y \in M$ such that $p = xy$ and $t = yx$.

Many of these invariants have an algebraic meaning. For example, (17) is the size of the conjugacy class of p when the magma is a group; (16) is the size of the filter generated by p when the magma is a regular semigroup; (13) is the number of square roots of p; (11) is the size of the centralizer of p; (8) and (9) are the number of idempotents in the left or right ideals generated by p; (5) and (6) are the number of elements in the left or right ideals generated by p; (1) deals with the periodicity of p; etc.

The intersection of the blocks induced by these invariants partitions the elements of the algebra, and by a straightforward argument on first order logic any endomorphism must preserve the blocks of this partition.

3.2.2. *Generating Set.* We can cut down the size of the search tree by focusing only on the images of the elements in a generating set. Some generating sets may be more suitable for finding automorphisms than others:

- A smaller generating set is preferred because it would require fewer trials in constructing an isomorphism.
- Generators from partition blocks with fewer elements are preferred since any homomorphism must send the elements of a block into itself.

Our algorithm satisfies both the constraints above and efficiently produces a generating set. It is depicted in Algorithm 12.

Algorithm 12 Constructing Efficient Generating Set

$submagma \leftarrow [\,]$
$generator \leftarrow [\,]$
$candidates \leftarrow$ magma elements sorted by ascending block size
while $submagma \neq magma$ **do**
  $generator \leftarrow generator \cup$ the element that increase the size of $submagma$ most, and if 2 elements increase the size by the same amount, take the one from the smaller block.
  $submagma \leftarrow$ submagma generated by $generator$
  $candidates \leftarrow candidates$ with elements in submagma removed
end while

3.2.3. *The Automorphism Algorithm.* To find the automorphisms we start by partitioning the Magma according to the invariant vectors of each element and then obtain an efficient generating set. We then need to find the images of the generating set taking into account that each element in the generating set can only map to elements having the same invariant vector. Then we extend this partial map to a full map and check that the homomorphism condition is satisfied.

We tested the algorithm on groups, loops, and quasigroups, obtaining the same results as yielded by the existing tools. For magmas of orders 2 and 3, and semigroups of orders 6 and 7 (as there are no general tools to compute automorphisms) we double check our results against the output of Mace4 [44].

3.2.4. *Algebras of Type (2^m,1^n).* Given an algebra of type (2^m,1^n) its automorphism group can be computed by taking the intersection of all the automorphism groups of the binary operations that also commute, function composition-wise, with all the unary operations.

A useful observation is that the intersection of the trivial group and any group is the trivial group. Therefore, if any of the automorphism group generated in the process is a trivial group, then the search can be terminated with the trivial group declared the automorphism group for the algebra.

This algorithm is implemented in the **CreamAutomorphisms** function in the CREAM package, as shown in Algorithm 13.

Algorithm 13 CreamAutomorphisms(algebra)

$binaryOperations \leftarrow AllBinaryOperations(algebra)$
$unaryOperations \leftarrow AllUnaryOperations(algebra)$
$D \leftarrow \emptyset$
repeat
  $B \leftarrow$ Pop a binary operation from $binaryOperations$
  $C \leftarrow$ all automorphisms of B
  $C \leftarrow$ all automorphisms in C that commute with every unary operation in $unaryOperations$
  if $C = \emptyset$ **then**
    return $\emptyset$
  end if
  $D \leftarrow D \cup C$
until $binaryOperations = \emptyset$
return Intersection(D)

3.3. **Endomorphisms algorithm.** The classic approach to calculate endomorphisms would be to use **Mace4** to search for endomorphisms and while this is very effective and fast for low order algebra further optimizations are needed for high order algebras in which the congruences of the algebra can be used to limit the **Mace4** search space for endomorphisms.

To compute all endomorphisms of high order algebras, CREAM does the following:

(1) Compute the congruences of the algebra A;
(2) For each congruence R, compute A/R (except for the trivial congruence);
(3) Compute the subalgebras of A that are isomorphic to A/R (using the finite model builder Mace4);
(4) For each compatible pair subalgebra/congruence derive a corresponding endomorphism;
(5) Add the automorphisms of A.

The congruences of A are calculated with the algorithms described in 3.1. For each congruence R (except for the trivial congruence that will be dealt with later), a representation of the A/R operation can be obtained by replacing the values in A's operation tables by their roots, and eliminating rows and columns not corresponding to roots. If we have the algebra:

```
[ [ [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ],
    [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ],[ 1, 1, 1, 2, 3, 1 ] ] ]
```

and the congruence

```
[ [ 1, 2, 3], [4], [5, 6] ] - [ -3, 1, 1, -1, -2, 5 ],
```

then the A/R operation is:

	1	4	5
1	1	1	1
4	1	1	1
5	1	1	1

and passed to Mace4 as follows:

```
f(a0,a0)=a0.
a0!=a3.
f(a0,a3)=a0.
a0!=a4.
f(a0,a4)=a0.
f(a3,a0)=a0.
f(a3,a3)=a0.
a3!=a4.
f(a3,a4)=a0.
f(a4,a0)=a0.
f(a4,a3)=a0.
f(a4,a4)=a0.
```

where f is the binary operation of the algebra. It is worth observing that Mace4 is zero-based, unlike GAP, which is one-based.

Running Mace4 with the definition of the algebra operation and the above encoding of A/R as assumptions we get 16 possible models each one of them corresponding to a different endomorphism of the algebra A. One of these models would be in Mace4's output:

```
interpretation( 6, [number = 1,seconds = 0], [
    function(a0, [0]),
    function(a3, [1]),
    function(a4, [2]),
    function(fa1(_,_), [
        0,0,0,0,0,0,
        0,0,0,0,0,0,
        0,0,0,0,0,0,
        0,0,0,0,0,0,
        0,0,0,0,0,0,
        0,0,0,1,2,0])]).
```

From this model we get the partial mapping $f : \{1,4,5\} \subseteq A \to \{1,2,3\}$, defined by $f(1) = 1$, $f(4) = 2$ and $f(5) = 3$, that can be represented by $[1,,,2,3,]$. The gaps can be easily filled since elements of the same block must map to the same image. In this example, 2 and 3 have the same image as 1, while 6 and 5 have the same image yielding the endomorphism: $[1,1,1,2,3,3]$.

Repeating this process for all congruences and for all models obtained with Mace4 from each congruence, all the algebra's endomorphisms (except for the automorphisms) can be obtained.

Finally, for the trivial congruence the associated endomorphisms would be also automorphisms and in this case it is more efficient to use the the algorithm described in 3.2 than using Mace4.

3.4. **Monomorphisms algorithms.** Invariants can be used to speed up the process of finding an monomorphism from one magma to another, or from one algebra to another. If f is an injective homomorphism from a magma A to a magma B, then it is a isomorphism from A to B restricted to the range of f. Thus, the same ideas of applying invariants in constructing automorphisms can be used for constructing monomorphisms. Specifically, an monomorphism f can map an element $a \in A$ to $b \in B$ only if a and b have the same invariant vector. This greatly reduces the search space for monomorphisms between A and B.

Algorithm 14 CreamAllMonomorphismMagmas(magma1, magma2)

$genL \leftarrow$ generating set from $magma1$
for i = 1 to $Size(magma2)$ **do**
  $rangeM[i] \leftarrow$ list of elements of $magma2$ that have same invariant vector as $genL[i]$
end for
$monoMaps \leftarrow \emptyset$
for all mappings m from $genL$ to elements of $rangeM$ s.t. $genL[i]$ maps only to elements in $rangeM[i]$ **do**
  if m is an injective homomorphism **then**
    $monoMaps \leftarrow monoMaps \cup m$
  end if
end for
return $monoMaps$

Algorithm 15 CreamAllMonomorphism(algebra1, algebra2)

b1 $\leftarrow$ first binary operation in $algebra1$
b2 $\leftarrow$ first binary operation in $algebra2$
$monoList \leftarrow$ CreamAllMonomorphismMagmas($b1, b2$)
$algebraMonoList \leftarrow []$
for each $monoMap$ in $monoList$ **do**
  $mono \leftarrow true$
  for each corresponding pair of operations $b1 \in algebra1, b2 \in algebra2$ **do**
    if $monoMap$ is not an isomorphism from $b1$ to $b2$ **then**
      $mono \leftarrow false$
      break out of for loop
    end if
  end for
  if $mono$ **then**
    Add $monoMap$ to $algebraMonoList$ if not already in the list
  end if
end for
return $algebraMonoList$

In this section only the function CreamAllMonomorphisms is addressed but the Cream package also includes the functions CreamExistsMonomorphism (returning true if a monomorphism exists) and CreamOneMonomorphism (returning one monomorphism).

3.5. **Epimorphisms algorithm.** CreamAllEpimorphisms(A_1, A_2) returns all epimorphisms from A_1 to A_2, provided the algebras are compatible. The algorithm first finds all congruences φ of A_1 such that A_1/φ and A_2 are isomorphic. For each such φ, the corresponding epimorphisms are obtained by composing the quotient map with an isomorphism to A_2 and all automorphisms of A_2.

For efficiency, a size check is implemented before searching for isomorphisms and the automorphisms of A_2 are only calculated once.

Algorithm 16 CreamAllEpimorphisms(algebra1, algebra2)

$allCongruences \leftarrow CreamAllCongruences(algebra1)$
$dimension1 \leftarrow SizeAlgebra(algebra1)$
$dimension2 \leftarrow SizeAlgebra(algebra2)$
$autoList \leftarrow []$
$epiList \leftarrow []$
for all $cong$ in $allConguences$ **do**
  if $dimension2 = CreamNumberOfBlocks(cong)$ **then**
    $[qalgebra, mapToQalgebra] \leftarrow QuotientAlgebraFromCongruence(algebra1, cong)$
    $iso \leftarrow IsomorphismAlgebras(qalgebra, algebra2)$
    if not $iso = fail$ **then**
      if autoList = [] **then**
        $autoList \leftarrow CreamAutomorphisms(algebra2)$
      end if
      for all $auto$ in $autoList$ **do**
        $epi \leftarrow []$
        for $j = 1$ to $dimension1$ **do**
          $epi[j] \leftarrow auto[iso[mapToQalgebra[i]]]$
        end for
        Add epi to $epiList$ if not already in the list
      end for
    end if
  end if
end for
return $epiList$

In this section only the function CreamAllEpimorphisms is addressed but the Cream package also includes the functions CreamExistsEpimorphism (returning true if a epimorphism exists) and CreamOneEpimorphism (returning one epimorphism).

3.6. **SubUniverses algorithm.** The SubUniverses algorithm returns a list of all underlying sets of all subalgebras of the input algebra. It first generates all 1-generated subalgebras, then iteratively expands each i-generated algebra by adding another generator.

For performance enhancement, these expansions are limited to one element from each orbit of the algebra's automorphism group. At the end of each cycle, the remaining $(i + 1)$-generated subuniverses are obtained by applying the automorphism group. The algorithm stops when an iteration produces only one subuniverse with the same size of the algebra.

The algorithm relies on the SubUniverseFromElement routine, which calculates the subalgebra generated by a subalgebra and an additional element. This routine assumes that the input subuniverse is closed under the algebra's operations. The algorithm adds the element to the updated subuniverse, calculates all directly generated additional elements, and places them in a temporary list. These elements are then filtered for those that are already in the subuniverse, and the remaining elements are put into a second list. Elements from the second list are then added to the subuniverse iteratively.

Algorithm 17 CreamAllSubUniverses(algebra)

$dimension \leftarrow CreamSizeAlgebra(algebra)$
$autoList \leftarrow CreamAutomorphisms(algebra)$
for $i = 1$ to $dimension$ **do**
  $sigma[i] \leftarrow []$
  if $i = 1$ **then**
    $sigmaMinus \leftarrow [[]]$
  else
    $sigmaMinus \leftarrow sigmaExpanded[i-1]$
  end if
  for all $cSigma$ in $sigmaMinus$ **do**
    $elemList \leftarrow [1..dimension]$
    Remove all elements in $cSigma$ from $elemList$
    while $Size(elemList) <> 0$ **do**
      $j \leftarrow elemList[1]$
      $sUniverse \leftarrow SubUniverseFromElement(algebra, cSigma, j)$
      Add $sUniverse$ to $sigma[i]$ if not already in the list
      $autoEs \leftarrow$ the orbit of j under $autoList$
      Remove all elements in $autoEs$ from $elemList$
    end while
  end for
  $sigmaExpanded[i] \leftarrow []$
  for all $sUniverse$ in $sigma[i]$ **do**
    $autoUniverses \leftarrow$ the orbit of $sUniverse$ under $autoList$
    Add all elements of $autoUniverses$ to $sigmaExpanded[i]$ if not already in the list
  end for
  if $Size(sigma[i]) = 1$ and $Size(sigma[i][1]) = dimension$ **then**
    $break$
  end if
end for
$sUniverses \leftarrow \cup_{j=1}^{i} sigmaExpended[j]$
return $sUniverses$

Algorithm 18 SubUniverseFromElement(algebra, initSubUniverse, element)

$dimension \leftarrow CreamSizeAlgebra(algebra)$
$elemList \leftarrow [element]$
$newSubUniverse \leftarrow initSubUniverse$
$nElem \leftarrow Size(elemList)$
while $nElem > 0$ **do**
  $currentElem \leftarrow elemList[nElem]$
  Remove element in position $nElem$ from $elemList$
  $newElemList \leftarrow []$
  for all operations op of the $algebra$ **do**
    if op is binary **then**
      Add $op[currentElem][currentElem]$ to $newElemList$ if not already in the list
      for all $subUniverseElem$ in $newSubUniverse$ **do**
        Add $op[currentElem][subUniverseElem]$ to $newElemList$ if not already in the list
        Add $op[subUniverseElem][currentElem]$ to $newElemList$ if not already in the list
      end for
    else
      Add $op[currentElem]$ to $newElemList$ if not already in the list
    end if
  end for
  Add $currentElem$ to $newSubUniverse$ if not already in the list
  $elemList \leftarrow elemList \cup (newElemList \setminus newSubUniverse)$
  $nElem \leftarrow Size(elements)$
  if $Size(newSubUniverse) + nElem = dimension$ **then**
    $newSubUniverse \leftarrow [1..dimension]$
    $nElem \leftarrow 0$
  end if
end while
return $newSubUniverse$

3.7. **DivisorUniverses algorithm.** CreamAllDivisorUniverses(A_1, A_2) checks if A_1 has a divisor that is isomorphic to A_2, where A_1 and A_2 are compatible algebras. Its return is a list of all pairs (B, φ), such that B is a subuniverse of A_1, φ is a congruence of the subalgebra B, and B/φ is isomorphic to A_2.

The algorithm first calculates the subalgebras of A_1, then their quotients and then checks those for isomorphisms to A_2, in each case using the corresponding algorithms of the Cream package. A size condition is used to prune unnecessary calculations.

Algorithm 19 CreamAllDivisorUniverses(algebra1, algebra2)

$dimension \leftarrow CreamSizeAlgebra(algebra2)$
$subUniverseList \leftarrow CreamSubUniverses(algebra1)$
$divisorList \leftarrow []$
for all $subUniverse$ in $subUniverseList$ **do**
  if $dimension <= Size(subUniverse)$ **then**
    $subAlgebra \leftarrow CreamSubUniverse2Algebra(algebra1, subUniverse)$
    $congList \leftarrow CreamAllCongruences(subAlgebra)$
    $rcongList \leftarrow []$
    for all $cong$ in $congList$ **do**
      if $dimension = CreamNumberOfBlocks(cong)$ **then**
        Add $cong$ to $rcongList$ if not already in the list
      end if
    end for
    for all $cong$ in $rcongList$ **do**
      $[qAlgebra, mapToQalgebra] \leftarrow QuotientAlgebraFromCongruence(subAlgebra, cong)$
      if $CreamAreAlgebrasIsomorphic(qAlgebra, algebra2)$ **then**
        Add $[subUniverse, cong]$ to $divisorList$ if not already in the list
      end if
    end for
  end if
end for
return $divisorList$

In this section only the function CreamAllDivisorUniverses is addressed but the Cream package also includes the functions CreamExistsDivisor (returning true if a divisor between the algebras exists) and CreamOneDivisorUniverse (returning one divisor between the algebras).

3.8. **DirectlyReducible algorithm.** CreamAllDirectlyReducible(A) looks for two (non-trivial) commuting congruences ϕ, ψ of A, whose meet is trivial and whose join is $A \times A$. If those exist, A is isomorphic to the direct product $A/\phi \times A/\psi$.

Because all algebras are finite, it suffices to check that the congruences have a trivial meet, and that the corresponding product algebra has the correct size. The algorithm uses numerical constraints to avoid unnecessary calculations.

The return of the algorithm is a list of pairs of congruences. These pairs should be consider as sets, i.e. if (α, β) is listed the (equally valid) pair (β, α) is not listed separately.

Algorithm 20 CreamAllDirectlyReducible(algebra)

$dimension \leftarrow CreamSizeAlgebra(algebra)$
$congList \leftarrow CreamAllCongruences(algebra)$
$pCongList \leftarrow []$
for all $cong$ in $congList$ **do**
  $cNBlocks \leftarrow CreamNumberOfBlocks(cong)$
  if $cNBlocks <> 1$ and $cNBlocks <> dimension$ and $IsInt(dimension/cNBlocks)$ **then**
    Add $cong$ to $pCongList$
  end if
end for
$pairs \leftarrow []$
for all 2-element subsets $\{i, j\}$ of $pCongsList$ **do**
  $iNBlocks \leftarrow CreamNumberOfBlocks(i)$
  $jNBlocks \leftarrow CreamNumberOfBlocks(j)$
  if $iNBlocks * jNBlocks = dimension$ **then**
    $cMeet \leftarrow MeetPartition(i, j)$
    $cMeetNBlocks \leftarrow CreamNumberOfBlocks(cMeet)$
    if $cMeetNBlocks = dimension$ **then**
      Add $[i, j]$ to $pairs$
    end if
  end if
end for
return $pairs$

In this section only the function CreamAllDirectlyReducible is addressed but the Cream package also includes the functions CreamExistsDirectlyReducible (returning true if there are commuting congruences R, T where A/R x A/T is isomorphic to A) and CreamOneDirectlyReducible (returning one pair of commuting congruences R, T where A/R x A/T is isomorphic to A).

4. Performance

4.1. **Congruences algorithm performance.** The **semigroup** GAP package [45] includes the function **CongruencesOfSemigroup** that returns the congruences of a semigroup, for some classes of semigroups. For Rees matrix semigroups the function **CongruencesOfSemigroup** takes advantage of theoretical results and hence achieves better results than **CreamAllCongruences**. But to achieve this kind of performance the semigroup needs to be created using the GAP functions **ReesMatrixSemigroup** or **ReesZeroMatrixSemigroup**. If instead an isomorphic copy of a Rees Matrix (Zero) semigroup is created using a multiplication table the performance will be much worse.

For Rees Matrix Semigroups or Rees Zero Matrix Semigroups (generated with **ReesMatrixSemigroup** or **ReesZeroMatrixSemigroup**) the function **CongruencesOfSemigroup** takes advantage of their particular structure by applying the efficient linked triple algorithm [35]. However, this efficiency can only be realized in this restricted setting.

UACalc also implements Freese's algorithm, and was done under his supervision in Java and Jython. It includes a GUI version in Java and a command line interface in Jython. The performance of the GUI version is poor and it will not be used for comparison. On the other hand the command line interface has a performance that is comparable to **CreamAllCongruences**. The same random Rees Matrix Semigroups that were used with **CongruencesOfSemigroup** and **CreamAllCongruences** were written into files readable by **UACalc** and the congruences were calculated in **UACalc**.

Table 1. Performance Comparison on calculating all congruences of Rees Matrix Semigroups.

Size	CongruencesOfSemigroup (ReesMatrixSemigroup) (ms)	CongruencesOfSemigroup (MultiplicationTable) (ms)	CreamAllCongruences (ms)	UACalc (ms)
12	1	15	< 1	1
18	2	28	< 1	3
24	5	55	2	7
30	1	92	4	32
36	4	139	10	34
42	11	203	16	55
48	8	296	24	77
54	5	416	46	138
60	5	553	60	220
66	1	671	82	291
72	9	873	136	409
78	2	1 098	180	555
84	5	1 341	233	743
90	4	1 622	307	961

CreamAllCongruences is consistently more than 3 times faster than **UACalc**. For these algebras with one binary operation the runtime rises roughly proportionally to n^4 (where n is the size of the algebra) or t^2 (being t the number cells of the multiplication matrix of the algebra operation).

4.2. **Automorphisms algorithm performance.** The CREAM package automorphism function was run on algebras of Type (2^m,1^n) to compare the timings against the Loops package. All experiments were run 7 times, with the lowest and highest times discarded. The averages of the remaining 5 are reported in Table 2. We see that the speeds from both packages are quite close.

Table 2. Performance Comparison between Loops and CREAM on Automorphism Group Generation.

Algebraic Structure	Loops Package(sec)	CREAM Package(sec)
Quasigroups, order 5	0.588	0.562
Quasigroups, order 6	520	541
Loops, order 6	0.101	0.086
Loops, order 7	14.623	13.945
Groups, order 32	0.854	0.935
Groups, order 64	39.98	41.05
Groups, order 128	12,031	12,160

4.3. **Endomorphisms algorithm performance.** To evaluate the performance of the algorithms to calculate endomorphisms, we tested both the classic algorithm and the congruence algorithm running with **Mace4**, as described in 3.3, on several types of Semigroups, Monoids and Groups. The only GAP function to calculate endomorphisms that came to our attention is the function Endomorphisms from the package SONATA. This function can calculate endomorphisms for a very narrow set of algebras, namely groups and near-rings. The groups in the list algebras were also run with SONATA.

Table 3. Performance Comparison between endomorphisms calculation algorithms

Algebra	Size	Number of Endomorphisms	Classic (ms)	Congruences (ms)	SONATA (ms)
GossipMonoid(3)	11	66	30	1 635	NA
PlanarPartitionMonoid(2)	14	72	56	233	NA
JonesMonoid(4)	14	72	45	232	NA
BrauerMonoid(3)	15	28	38	160	NA
PartitionMonoid(2)	15	89	50	313	NA
FullPBRMonoid(1)	16	1 426	585	5 134	NA
SymmetricGroup(4)	24	58	101	142	166
FullTransformationSemigroup(3)	27	40	116	364	NA
FullTransformationMonoid(3)	27	40	112	360	NA
SymmetricInverseMonoid(3)	34	54	274	504	NA
JonesMonoid(5)	42	113	746	777	NA
MotzkinMonoid(3)	51	98	1 972	2 400	NA
PartialTransformationMonoid(3)	64	138	2 586	1 963	NA
PartialBrauerMonoid(3)	76	165	7 897	7 796	NA
BrauerMonoid(4)	105	274	50 164	19 875	NA
SymmetricGroup(5)	120	146	32 861	2 979	201
PlanarPartitionMonoid(3)	132	393	95 460	25 803	NA
JonesMonoid(6)	132	393	131 472	22 889	NA
PartitionMonoid(3)	203	687	741 441	105 421	NA
SymmetricInverseMonoid(4)	209	282	470 901	75 940	NA

In this test the classic algorithm not using congruences is faster than using congruences for algebras up to size 60 but for larger algebras its runtime raises very fast and the algorithm using congruences becomes faster.

In view of these results, the **CreamEndomorphisms** function uses the classic algorithm for algebras with size up to 60 and the congruences algorithm for larger algebras.

The SONATA package achieves very good results for groups but it only works on a very narrow set of algebras.

5. Conclusion

The CREAM package provides efficient algorithms for the calculation of congruences, automorphisms and endomorphisms. These algorithms work with algebras of type $(2^m, 1^n)$, covering a very wide set of algebras and being consistently fast.

The integration with Mace4 allowed us to combine efficient algorithms such as [28] with the wide set of possibilities provided by a first order axiom-based model searcher.

The appendices of this article include a detailed performance discussion of the main algorithms and applications of the CREAM package to different algebra classes.

The CREAM package can be found and downloaded from `https://gitlab.com/rmbper/cream` and the outputs from the test runs described in section B can be found in `https://gitlab.com/rmbper/cream_dat`

Appendices

A. Detailed Performance

A.1. **Congruences algorithm performance.** The **semigroup** GAP package [45] includes the function **CongruencesOfSemigroup** that returns the congruences of a semigroup. This function doesn't work for all semigroups, but for those in which it works we have run the **CongruencesOfSemigroup** function against our **CreamAllCongruences** 5 times and calculated the average runtime after removing the best and worst runtimes getting the following results:

Table 4. Performance Comparison between **CongruencesOfSemigroup** and CREAM on calculating all congruences.

Size	Number of Algebras	CongruencesOfSemigroup(ms)	CreamAllCongruences(ms)
1	1	7	< 1
2	3	20	< 1
3	4	29	< 1
4	7	41	< 1
5	9	53	< 1
6	8	71	< 1
7	12	153	4
8	14	484	35
1-8	58	858	40

The number of semigroups for which **CongruencesOfSemigroup** works is very limited, yet even for those algebras **CreamAllCongruences** is more than 20 times faster.

The tests show that there is a very specific type of semigroup in which the function **CongruencesOfSemigroup** takes advantage of theoretical results and hence achieves better results than **CreamAllCongruences**. These are the Rees Matrix Semigroups or the Rees Zero Matrix Semigroups. But to achieve this kind of performance the semigroup needs to be created using the GAP functions **ReesMatrixSemigroup** or **ReesZeroMatrixSemigroup**. If instead an isomorphic copy of a Rees Matrix (Zero) semigroup is created using a multiplication table the performance will be similar to what was found above. See in the following table the average runtime for 10 runs with random Rees Matrix Semigroups: For Rees Matrix Semigroups or Rees Zero Matrix Semigroups (generated with **ReesMatrixSemigroup** or **ReesZeroMatrixSemigroup**) the function **CongruencesOfSemigroup** takes advantage of their particular structure by applying the efficient linked triple algorithm [35]. However, this efficiency can only be realized in this restricted setting.

UACalc also implements Freese's algorithm, and was done under his supervision in Java and Jython. It includes a GUI version in Java and a command line interface in Jython. The performance of the GUI version is poor and it will not be used for comparison. On the other hand the command line interface has a performance that is comparable to **CreamAllCongruences**. The same random Rees Matrix Semigroups that were used with **CongruencesOfSemigroup** and **CreamAllCongruences** were written into files readable by **UACalc** and the congruences were calculated in **UACalc**. **CreamAllCongruences** is consistently more than 3 times faster than **UACalc**. For these algebras with one binary operation the runtime rises roughly proportionally to n^4 (where n is the size of the algebra) or t^2 (being t the number cells of the multiplication matrix of the algebra operation).

To further compare the performance of **CreamAllCongruences** with **UACalc**, tests with several specific types of Semigroups, Monoids and Groups were run.

Table 5. Performance Comparison on calculating all congruences of Rees Matrix Semigroup.

Size	CongruencesOfSemigroup (ReesMatrixSemigroup) (ms)	CongruencesOfSemigroup (MultiplicationTable) (ms)	CreamAllCongruences (ms)	UACalc (ms)
12	1	15	< 1	1
18	2	28	< 1	3
24	5	55	2	7
30	1	92	4	32
36	4	139	10	34
42	11	203	16	55
48	8	296	24	77
54	5	416	46	138
60	5	553	60	220
66	1	671	82	291
72	9	873	136	409
78	2	1 098	180	555
84	5	1 341	233	743
90	4	1 622	307	961

Table 6. Performance Comparison between CREAM and UACalc

Algebra	Size	Number of Congruences	CreamAllCongruences (ms)	UACalc (ms)
GossipMonoid(3)	11	84	1	2
PlanarPartitionMonoid(2)	14	9	1	1
JonesMonoid(4)	14	9	< 1	1
BrauerMonoid(3)	15	7	< 1	2
PartitionMonoid(2)	15	13	< 1	1
FullPBRMonoid(1)	16	167	3	4
SymmetricGroup(4)	24	4	3	10
FullTransformationSemigroup(3)	27	7	3	14
FullTransformationMonoid(3)	27	7	3	7
SymmetricInverseMonoid(3)	34	7	6	25
JonesMonoid(5)	42	6	12	36
MotzkinMonoid(3)	51	10	23	73
PartialTransformationMonoid(3)	64	7	63	223
PartialBrauerMonoid(3)	76	16	117	351
BrauerMonoid(4)	105	19	430	1 420
SymmetricGroup(5)	120	3	812	2 700
PlanarPartitionMonoid(3)	132	10	1 094	3 490
JonesMonoid(6)	132	10	950	3 444
PartitionMonoid(3)	203	16	6 500	19 979
SymmetricInverseMonoid(4)	209	11	6 970	22 318
FullTransformationSemigroup(4)	256	11	22 529	53 557
FullTransformationMonoid(4)	256	11	22 385	52 361
MotzkinMonoid(4)	323	11	47 593	134 945
JonesMonoid(7)	429	7	161 987	485 004

The performance for these algebras is mostly similar to what was found for the Rees Matrix Semigroups, with a bigger variance in the results. On average **CreamAllCongruences** is between 3 and 3,5 times faster

than **UACalc** but, depending on the algebras, we get improvements spanning from 2 times to 4,5 times. The runtime also rises roughly proportionally to n^4 (although with more variance).

A.2. **Automorphisms algorithm performance.** The CREAM package automorphism function was run on algebras of Type (2^m,1^n) to compare the timings against the Loops package. All experiments were run 7 times, with the lowest and highest times discarded. The averages of the remaining 5 are reported in Table 7. We see that the speeds from both packages are quite close.

Table 7. Performance Comparison between Loops and CREAM on Automorphism Group Generation.

Algebraic Structure	Loops Package(sec)	CREAM Package(sec)
Quasigroups, order 5	0.588	0.562
Quasigroups, order 6	520	541
Loops, order 6	0.101	0.086
Loops, order 7	14.623	13.945
Groups, order 32	0.854	0.935
Groups, order 64	39.98	41.05
Groups, order 128	12,031	12,160

We have also run experiments on generating automorphism groups for magmas and semigroups. Results are displayed on Table 8. For these experiments, we do not have results from other GAP packages to compare with. Therefore the following results can be taken as the benchmark to beat in future improvements.

Table 8. Performance of CREAM on Automorphism Group Generation for Magmas and Semigroups.

Algebraic Structure	Time(sec)
Magmas, order 2	0.016
Magmas, order 3	1.253
Semigroups, order 6	2.316
Semigroups, order 7	45.947

The CREAMAutomorphisms runs substantially faster than the simple intersection of automorphism groups, especially when the automorphism group is the trivial group. Sample results are shown in Table 9.

Table 9. Performance Comparison between CREAMAutomorphisms and Intersection of Automorphism Groups.

Algebra of type (2, 2, 2, ...)	Algebra AutomorphismGroup (sec)	Intersection of Automorphism Groups (sec)	Is Trival Group the Automorphism Group?
All 2,328 non-isomorphic groups of order 128	153.8	201.5	No
All 67 non-isomorphic groups of order 243	10.3	56.3	No
All 15 non-isomorphic groups of order 625	11.3	122.9	No
All 15 non-isomorphic groups of order 999	2.1	168.9	Yes

A.3. **Endomorphisms algorithm performance.** To evaluate the performance of the algorithms to calculate endomorphisms, we tested both the classic algorithm and the congruence algorithm running with **Mace4**, as described in 3.3, on several types of Semigroups, Monoids and Groups. The only GAP function to calculate endomorphisms that came to our attention is the function Endomorphisms from the package SONATA. This function can calculate endomorphisms for a very narrow set of algebras, namely groups and near-rings. The groups in the list algebras were also run with SONATA.

Table 10. Performance Comparison between endomorphisms calculation algorithms

Algebra	Size	Number of Endomorphisms	Classic (ms)	Congruences (ms)	SONATA (ms)
GossipMonoid(3)	11	66	30	1 635	NA
PlanarPartitionMonoid(2)	14	72	56	233	NA
JonesMonoid(4)	14	72	45	232	NA
BrauerMonoid(3)	15	28	38	160	NA
PartitionMonoid(2)	15	89	50	313	NA
FullPBRMonoid(1)	16	1 426	585	5 134	NA
SymmetricGroup(4)	24	58	101	142	166
FullTransformationSemigroup(3)	27	40	116	364	NA
FullTransformationMonoid(3)	27	40	112	360	NA
SymmetricInverseMonoid(3)	34	54	274	504	NA
JonesMonoid(5)	42	113	746	777	NA
MotzkinMonoid(3)	51	98	1 972	2 400	NA
PartialTransformationMonoid(3)	64	138	2 586	1 963	NA
PartialBrauerMonoid(3)	76	165	7 897	7 796	NA
BrauerMonoid(4)	105	274	50 164	19 875	NA
SymmetricGroup(5)	120	146	32 861	2 979	201
PlanarPartitionMonoid(3)	132	393	95 460	25 803	NA
JonesMonoid(6)	132	393	131 472	22 889	NA
PartitionMonoid(3)	203	687	741 441	105 421	NA
SymmetricInverseMonoid(4)	209	282	470 901	75 940	NA

In this test the classic algorithm not using congruences is faster than using congruences for algebras up to size 60 but for larger algebras its runtime raises very fast and the algorithm using congruences becomes faster.

In view of these results, the **CreamEndomorphisms** function uses the classic algorithm for algebras with size up to 60 and the congruences algorithm for larger algebras.

B. APPLICATIONS

B.1. **Monolithic Algebras.** Some examples of the use of the package to calculate whether an algebra is monolithic can be seen next:

```
gap> algebra := [RecoverMultiplicationTable(6,1)];
[ [ [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ],
      [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ] ] ]
gap> CreamAllPrincipalCongruences(algebra);
[ [ -1, -1, -1, -1, -2, 5 ], [ -1, -1, -1, -2, 4, -1 ],
  [ -1, -1, -1, -2, -1, 4 ], [ -1, -1, -2, 3, -1, -1 ],
  [ -1, -1, -2, -1, 3, -1 ], [ -1, -1, -2, -1, -1, 3 ],
  [ -1, -2, 2, -1, -1, -1 ], [ -1, -2, -1, 2, -1, -1 ],
  [ -1, -2, -1, -1, 2, -1 ], [ -1, -2, -1, -1, -1, 2 ],
  [ -2, 1, -1, -1, -1, -1 ], [ -2, -1, 1, -1, -1, -1 ],
  [ -2, -1, -1, 1, -1, -1 ], [ -2, -1, -1, -1, 1, -1 ],
  [ -2, -1, -1, -1, -1, 1 ] ]
gap> CreamIsAlgebraMonolithic(algebra);
false
gap> algebra := [RecoverMultiplicationTable(6,19)];
[ [ [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1 ],
      [ 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 2, 1, 1 ], [ 1, 1, 2, 1, 1, 1 ] ] ]
gap> CreamAllPrincipalCongruences(algebra);
[ [ -2, 1, -1, -1, -1, -1 ], [ -2, 1, -1, -1, -2, 5 ],
  [ -2, 1, -1, -2, 4, -1 ], [ -2, 1, -1, -2, -1, 4 ],
  [ -2, 1, -2, 3, -1, -1 ], [ -2, 1, -2, -1, 3, -1 ],
  [ -2, 1, -2, -1, -1, 3 ], [ -3, 1, 1, -1, -1, -1 ],
  [ -3, 1, -1, 1, -1, -1 ], [ -3, 1, -1, -1, 1, -1 ],
  [ -3, 1, -1, -1, -1, 1 ] ]
gap> CreamIsAlgebraMonolithic(algebra);
true
```

B.2. **Small Semigroups.** For all small semigroups up to size 6, all its congruences and endomorphisms were calculated. It was moreover determined whether the semigroups were monolithic (this was calculated up to size 8).

Table 11. Determination of all monolithic semigroups up to size 8

Size	Number of semigroups	Number of monoliths	Performance (ms)
1	1	0	0
2	4	4	0
3	18	7	0
4	126	16	8
5	1 160	103	108
6	15 973	1 823	1 712
7	836 021	149 020	127 356
8	1 843 120 128	48 438 046	462 897 348

All automorphisms for semigroups up to size 7 were also calculated separately and the automorphism group and its ID was calculated for each of these semigroups. The group ID is the identification of a group in the Small Group GAP library [20].

The CREAM library doesn't calculate the automorphisms of an algebra as an automorphism group but as a list of bijective mappings, but there is an easy way to calculate the automorphism group from the list of mappings. To do this, all the mappings are converted into permutations using the **PermList** function and each of these permutations is used as generator for the group. Having this we can get the Group Id using the **IdGroup** function.

The following table gives the automorphism of semigroups up to size 7 (and reproduces the results in [9]):

Table 12. Computation of the automorphism groups of semigroups up to size 7

Size	2	3	4	5	6	7
# Semigroups	4	18	126	1 160	15 973	836 021
[1, 1] trivial	3	12	78	746	10 965	746 277
[2, 1] C_2	1	5	39	342	4 121	76 704
[3, 1] C_3				2	26	412
[4, 1] C_4				1	7	82
[4, 2] $C_2 \times C_2$			3	26	441	7 314
[5, 1] C_5						6
[6, 1] S_3		1	5	33	300	3 638
[6, 2] C_6						37
[8, 2] $C_4 \times C_2$						4
[8, 3] D_8				1	17	169
[8, 5] $C_2 \times C_2 \times C_2$					6	172
[10, 1] D_{10}						2
[12, 4] D_{12}				4	49	790
[16, 11] $C_2 \times D_8$						10
[24, 12] S_4			1	4	30	277
[24, 14] $C_2 \times C_2 \times S_3$						14
[36, 10] $S_3 \times S_3$					2	24
[48, 48] $C_2 \times S_4$					4	45
[72, 40] $(S_3 \times S_3) \wr C_2$						1
[120, 34] S_5				1	4	30
[144, 183] $S_3 \times S_4$						4
[240, 189] $C_2 \times S_5$						4
[720, 763] S_6					1	4
[5040, -] S_7						1

B.3. **Small Groups.** Using the Small Group GAP library [20] the groups from order 2 to 96 were extracted and the Cream functions were used with them:

- Order 2-31 - CreamAllCongruences, CreamAllEndomorphisms and CreamIsMonolithic
- Order 32-96 - CreamAllCongruences and CreamIsMonolithic

A summary from these runs is provided in Table 13.

Table 13. Summary for Small Group Runs

Size	Number of Groups	Number of Monolithic
2-7	8	6
8-15	19	9
16	14	6
17-23	17	7
24	15	2
25-31	19	7
32	50	16
33-47	54	10
48	52	4
49-63	69	16
64	52	22
65-71	17	3
72	50	3
73-79	18	5
80	52	1
81-95	70	13
96	231	13

B.4. **Groups, Semigroups and Monoids.** For a selection of larger groups, semigroups and monoids, all congruences and endomorphisms were calculated. It was also calculated whether these algebras were monolithic. A summary from these runs is provided in Table 14.

Table 14. # of Congruences and Endomorphisms for a selection of larger algebras

Algebra	Size	Number of Congruences	Number of Endomorphisms	Is Monolithic
GossipMonoid(3)	11	84	66	No
PlanarPartitionMonoid(2)	14	9	72	No
JonesMonoid(4)	14	9	72	No
BrauerMonoid(3)	15	7	28	No
PartitionMonoid(2)	15	13	89	No
FullPBRMonoid(1)	16	167	1426	No
SymmetricGroup(4)	24	4	58	Yes
FullTransformationSemigroup(3)	27	7	40	Yes
FullTransformationMonoid(3)	27	7	40	Yes
SymmetricInverseMonoid(3)	34	7	54	Yes
JonesMonoid(5)	42	6	113	No
MotzkinMonoid(3)	51	10	98	No
PartialTransformationMonoid(3)	64	7	138	Yes
PartialBrauerMonoid(3)	76	16	165	No
BrauerMonoid(4)	105	19	274	No
SymmetricGroup(5)	120	3	146	Yes
PlanarPartitionMonoid(3)	132	10	393	No
JonesMonoid(6)	132	10	393	No
PartitionMonoid(3)	203	16	687	No
SymmetricInverseMonoid(4)	209	11	282	Yes
FullTransformationSemigroup(4)	256	11	345	Yes
FullTransformationMonoid(4)	256	11	345	Yes

B.5. **Automorphisms of Specific Algebra Classes.** The following table contains the groups that appear as automorphims groups of associative rings up to order 15.

Table 15. Computation of the automorphism groups of rings up to size 15

Size	2	3	4	5	6	7	8	9	10	11	12	13	14	15
# Rings	2	2	11	2	4	2	52	11	4	2	22	2	4	4
[1, 1] trivial	2	1	3		2		4				3			
[2, 1] C_2		1	6	1		1	12	1	2		4		2	1
[4, 1] C_4										1				
[4, 2] $C_2 \times C_2$					1		6	3				1		
[6, 1] S_3			2				1							
[6, 2] C_6							1							
[8, 3] D_8							3							
[8, 5] $C_2 \times C_2 \times C_2$							4	1			2			
[12, 4] D_{12}								2						
[16, 11] $C_2 \times D_8$							3							
[16, 14] $C_2 \times C_2 \times C_2 \times C_2$											2			
[24, 12] S_4					1									
[24, 13] $C_2 \times A_4 \times S_3$							2							
[32, 46] $C_2 \times C_2 \times D_8$											4			
[36, 10] $S_3 \times S_3$							4							
[48, 48] $C_2 \times S_4$							7							
[64, 261] $C_2 \times C_2 \times C_2 \times D_8$														1
[72, 40] $(S_3 \times S_3) \wr C_2$								2						
[120, 34] S_5					1									
[144, 183] $S_3 \times S_4$							2							
[216, 162] $S_3 \times S_3 \times S_3$											2			
[576, 8653] $S_4 \times S_4$									1					
[720, 763] S_6						1								
[5040, -] S_7							3							
[13824, -] $S_4 \times S_4 \times S_4$														1
[14400, -] $S_5 \times S_5$											1			
[17280, -] $S_6 \times S_4$											2			
[40320, -] S_8								2						
[362880, -] S_{10}									1	1				
[518400, -] $S_6 \times S_6$													1	
[39916800, -] S_{11}											2			
[479001600, -] S_{12}												1		
[6227020800, -] S_{13}													1	
[87178291200, -] S_{14}														1

The variety of *Quasi-MV-algebras* [37] is defined by the following identities (in ProverX syntax [13,17]):

```
(x + z) + y = x + (y + z).
x'' = x.
x + 1 = 1.
(x' + y)' + y = (y' + x)' + x.
(x + 0)' = x' + 0.
(x + y) + 0 = x + y.
0' = 1.
```

Table 16. Computation of the automorphism groups of Quasi-MV-algebras of size up to 12

Size	2	3	4	5	6	7	8	9	10	11	12
# Quasi-MV-algebras	1	1	4	4	11	11	27	27	60	62	131
[1, 1] trivial	1	1	3	2	7	4	16	8	35	17	76
[2, 1] C_2			1	2	3	4	5	9	10	18	19
[4, 2] $C_2 \times C_2$						2	1	4	2	10	5
[6, 1] S_3					1		3		5		9
[8, 5] $C_2 \times C_2 \times C_2$								2	1	4	2
[12, 4] D_{12}							1		2		5
[16, 14] $C_2 \times C_2 \times C_2 \times C_2$										2	1
[24, 12] S_4						1		2		4	
[24, 14] $C_2 \times C_2 \times C_3$									1		2
[48, 48] $C_2 \times S_4$								1		2	
[48, 51] $C_2 \times C_2 \times C_2 \times S_3$											1
[96, 226] $C_2 \times C_2 \times S_4$										1	
[120, 34] S_5							1	1	2		4
[240, 189] $C_2 \times S_5$									1		2
[480, 1186] $C_2 \times C_2 \times S_5$											1
[720, 763] S_6										2	
[1440, 5842] $C_2 \times S_6$										1	
[5040, -] S_7									1		2
[10080, -] $C_2 \times S_7$											1
[40320, -] S_8										1	
[362880, -] S_9											1

The quasivariety of *BCI algebras* [2] is defined by the following identities (in ProverX syntax [13, 17]):

```
((x * y) * (x * z)) * (z * y) = 0.
(x * (x * y)) * y = 0.
x * x = 0.
((x * y = 0) & (y * x = 0)) -> (x = y).
(x * 0 = 0) -> (x = 0).
```

Table 17. Computation of the automorphism groups of BCI algebras of size up to 7

Size	2	3	4	5	6	7
# BCI algebras	2	5	22	118	974	10,834
[1, 1] trivial	2	3	13	78	679	7,970
[2, 1] C_2		2	7	31	241	2,384
[4, 1] C_4				1	1	2
[4, 2] $C_2 \times C_2$				2	20	207
[6, 1] S_3			2	5	25	195
[6, 2] C_6						1
[8, 2] $C_4 \times C_2$						1
[8, 3] D_8						4
[8, 5] $C_2 \times C_2 \times C_2$						3
[12, 4] D_{12}					4	35
[24, 12] S_4				1	4	22
[36, 10] $S_3 \times S_3$						2
[48, 48] $C_2 \times S_4$						3
[120, 34] S_5					1	4
[720, 763] S_6						1

The variety of *quandles* [43] is defined by the following identities (in ProverX syntax [13, 17]):

```
x v (y v z) = (x v y) v (x v z).
(x ^ y) ^ z = (x ^ z) ^ (y ^ z).
(x v y) ^ x = y.
x v (y ^ x) = y.
x v x = x.
```

Table 18. Computation of the automorphism groups of quandles of size up to 6

Size	2	3	4	5	6
# quandles	1	3	7	22	73
[2, 1] C_2	1	1	1		
[3, 1] C_3			1	1	
[4, 1] C_4				1	2
[4, 2] $C_2 \times C_2$			1	5	8
[5, 1] C_5					1
[6, 1] S_3		2	1	2	3
[6, 2] C_6				2	15
[8, 2] $C_4 \times C_2$					2
[8, 3] D_8			1	3	8
[8, 5] $C_2 \times C_2 \times C_2$					7
[12, 3] A_4			1	1	
[12, 4] D_{12}				3	6
[16, 11] $C_2 \times D_8$					5
[18, 3] $C_3 \times S_3$					3
[20, 3] $C_5 \times C_4$				3	3
[24, 12] S_4			1		2
[24, 13] $C_2 \times A_4 \times S_3$					3
[36, 10] $S_3 \times S_3$					1
[48, 48] $C_2 \times S_4$					2
[72, 40] $(S_3 \times S_3) \wr C_2$					1
[120, 34] S_5				1	
[720, 763] S_6					1

B.6. **Algebra Classes.** The CREAM package allows for the generation of lists of algebras of specific types based on its axiomatic definition. This is done using the function **CreamAlgebrasFromAxioms** that uses Mace4 to generate algebras according with this axiomatic definition. This function takes as input a string with the axioms in Mace4 format and a positive integer with the size of the generated algebras. The generated algebras can then be used with the CREAM functions that calculate congruences, endomorphisms and whether the algebra is monolithic. CREAM was applied in this way to the following classes of algebras:

- Almost distributive lattices
- BCI-algebras
- Bilattices
- Boolean algebras
- Boolean rings
- Commutative lattice-ordered monoids
- Commutative regular rings
- Complemented distributive lattices
- Complemented modular lattices
- Distributive lattice ordered semigroups
- Ockham algebras
- Ortholattices
- Orthomodular lattices
- Idempotent semirings
- Extra loops
- Digroups
- Commutative dimonoids

A summary from these runs is provided in Table 19.

Table 19. Summary for Algebra Class Runs

Algebra Class	Size	Algebra Type	Number of Algebras	Number of Monolithic
Almost distributive lattices	4	(2^2)	2	0
BCI-algebras	5	(2^1)	118	80
Bilattice	6	$(2^4, 1^1)$	32	32
Boolean algebra	8	$(2^2, 1^1)$	1	0
Boolean algebra	16	$(2^2, 1^1)$	1	0
Boolean ring	16	(2^2)	1	0
Commutative lattice-ordered monoids	5	(2^3)	199	97
Commutative regular rings	6	$(2^2, 1^2)$	72	66
Complemented distributive lattices	16	$(2^2, 1^1)$	1	0
Complemented distributive lattices	32	$(2^2, 1^1)$	1	0
Complemented modular lattices	8	$(2^2, 1^1)$	41	40
Distributive lattice ordered semigroups	4	(2^3)	479	170
Ockham algebras	6	$(2^2, 1^1)$	197	20
Ortholattices	7	$(2^2, 1^1)$	46	12
Orthomodular lattices	14	$(2^2, 1^1)$	33	31
Idempotent semiring	5	(2^2)	149	42
Extra loop	10	(2^3)	2	1
Digroup	8	$(2^2, 1^1)$	10	3
Commutative dimonoid	4	(2^2)	101	37

REFERENCES

[1] J. André, J. Araújo, P. J. Cameron, The classification of partition homogeneous groups with applications to semigroup theory. *J. Algebra* **452** (2016), 288–310.

[2] Y. Arai, K. Iséki and S. Tanaka, Characterizations of BCI, BCK-algebras, *Proc. Japan Acad.* (1966), **42** (2), 105–107.

[3] J. Araújo, J.P. Araújo, W. Bentz, P. J. Cameron, P. Spiga, A transversal property for permutation groups motivated by partial transformations. *J. Algebra* **573** (2021), 741–759.

[4] J. Araújo, W. Bentz, P. J. Cameron, The existential transversal property: a generalization of homogeneity and its impact on semigroups. *Trans. Amer. Math. Soc.* **374** (2021), no. 2, 1155–1195.

[5] J. Araújo, W. Bentz, P. J. Cameron, Primitive permutation groups and strongly factorizable transformation semigroups. *J. Algebra* **565** (2021), 513–530.

[6] J. Araújo, W. Bentz, P. J. Cameron, Orbits of primitive k-homogenous groups on (n?k)-partitions with applications to semigroups. *Trans. Amer. Math. Soc. 371* (2019), no. 1, 105–136.

[7] J. Araújo, W. Bentz, P. J. Cameron, G. Royle, A. Schaefer, Primitive groups, graph endomorphisms and synchronization. *Proc. Lond. Math. Soc.* (3) **113** (2016), no. 6, 829–867.

[8] J. Araújo, W. Bentz, E. Dobson, J. Konieczny, J. Morris, Automorphism groups of circulant digraphs with applications to semigroup theory. *Combinatorica* **38** (2018), no. 1, 1–28.

[9] J. Araújo, P.V. Bünau, J.D. Mitchell, M. Neunhöffer, Computing automorphisms of semigroups, J. Symbolic Comput., 45 (3) (2010), pp. 373-392

[10] J. Araújo, P. J. Cameron, B. Steinberg, Between primitive and 2-transitive: synchronization and its friends. *EMS Surv. Math. Sci.* **4** (2017), no. 2, 101–184.

[11] J. Araújo, P. J. Cameron, Two generalizations of homogeneity in groups with applications to regular semigroups. *Trans. Amer. Math. Soc.* **368** (2016), no. 2, 1159–1188.

[12] J. Araújo, P. J. Cameron, Primitive groups synchronize non-uniform maps of extreme ranks. *J. Combin. Theory Ser. B* **106** (2014), 98–114.

[13] J. Araújo, M. Kinyon and Yves Robert, Varieties of regular semigroups with uniquely defined inversion. Port. Math. 76 (2019), 205–228.

[14] J. Araújo and J. Konieczny, Automorphism groups of centralizers of idempotents, *J. Algebra* **269** (2003), 227–239.

[15] J. Araújo, and J. Konieczny, A Method of Finding Automorphism Groups of Endomorphism Monoids of Relational Systems, *Discrete Math.* **307** (2007) 1609–1620.

[16] J. Araújo and J. Konieczny, Automorphisms of Endomorphism Monoids of Relatively Free Bands, *Proc. Edinburgh Math. Soc.* **50** (2007) 1–21

[17] J. Araújo and Yves Robert, The System ProverX, www.proverx.com.

[18] E. Artin, "Geometric Algebra," Interscience, New York, 1957.

[19] M. V. Berlinkov, R. Ferens, M. Szykua, Preimage problems for deterministic finite automata. *J. Comput. System Sci.* **115** (2021), 214–234.

[20] Hans Ulrich Besche, Bettina Eick & Eamonn O'Brien, *GAP package SmallGrp - The GAP Small Groups Library, Version 1.4.2*; 2020, `https://www.gap-system.org/Packages/smallgrp.html`.

[21] Stanley N. Burris & H.P. Sankappanavar, *A Course in Universal Algebra*; 1981, `http://www.math.uwaterloo.ca/~snburris/htdocs/UALG/univ-algebra2012.pdf`.

[22] P.J. Cameron, Automorphism groups of graphs, "Selected Topics in Graph Theory. 2," 89–127, Academic Press, London, 1983.

[23] J. J. Cannon and D. Holt, Automorphism group computation and isomorphism testing in finite groups, *J. Symbolic Comput.* **35** (2003), 241–267.

[24] David Clark and Brian Davey, *Natural dualities for the working algebraist*, Cambridge Studies in Advanced Mathematics 57, 1998

[25] Andreas Distler & James Mitchel, *GAP package Smallsemi - A library of small semigroups, Version 0.6.12*; 2019, `https://www.gap-system.org/Packages/smallsemi.html`.

[26] S. P. Fitzpatrick and J. S. Symons, Automorphisms of transformation semigroups, *Proc. Edinburgh Math. Soc.* **19** (1974/75), 327-329.

[27] Ralph Freese, *UACalc, A Universal Algebra Calculator*; 2015, `http://www.uacalc.org/`.

[28] Ralph Freese, Computing congruences efficiently, *Algebra Universalis* **59(3)** (2008), 337–343. DOI:

[29] Ralph Freese, and Ralph McKenzie, *Commutator Theory for congruence modular varieties*, London Mathematical Society Lecture Notes, Vol. 125, 1987

[30] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.11.0*; 2020, `https://www.gap-system.org`.

[31] O. Garcia and Walter Taylor, *The lattice of interpretability types of varieties*, Memoirs of the American Mathematical Society, Vol. 50, no. 305, 1984

[32] L. M. Gluskĭn, Semigroups and rings of endomorphisms of linear spaces I, *Amer. Math. Soc. Transl.* **45** (1965), 105–137.
[33] J. Hagemmann and C. Herrmann, A concrete ideal multiplication for algebraic systems and its relation to congruence distributivity, *Arch. Math.* **32**, 234-245, 1979
[34] David Hobby and Ralph McKenzie, *The structure of finite algebras*, Contemporary Mathematics, Vol. 76, 1988
[35] J. M. Howie, *Fundamentals of Semigroup Theory*, Oxford University Press, New York, 1995.
[36] F. Klein, "Vergleichende Betrachtungen über neuere geometrische Forschungen," Andreas Diechert, Erlanger, 1872.
[37] A. Ledda, M. Konig, F. Paoli, and R. Giuntini. MV-algebras and quantum computation. *Studia Logica* (2006) **82** (2):245–270.
[38] I. Levi, Automorphisms of normal transformation semigroups, *Proc. Edinburgh Math. Soc. (2)* **28** (1985), 185–205.
[39] I. Levi, Automorphisms of normal partial transformation semigroups, *Glasgow Math. J.* **29** (1987), 149–157.
[40] A. E. Liber, On symmetric generalized groups, *Mat. Sbornik N. S.* **33** (1953), 531–544. (Russian)
[41] K. D. Magill, Semigroup structures for families of functions, I. Some homomorphism theorems, *J. Austral. Math. Soc.* **7** (1967), 81–94.
[42] A. I. Mal'cev, Symmetric groupoids, *Mat. Sbornik N.S.* **31** (1952), 136–151. (Russian)
[43] S. Matveev, Distributive groupoids in knot theory. *Math. USSR Sbornik* (1984) **47**, 73–83.
[44] W. McCune, *Mace4, Version 2009-02A*; 2009 `https://www.cs.unm.edu/~mccune/prover9/manual/2009-02A/mace4.html`.
[45] J. D. Mitchell and others, Semigroups - GAP package, Version 2.6, (2015). `https://gap-packages.github.io/Semigroups/`
[46] V. A. Molchanov, Semigroups of mappings on graphs, *Semigroup Forum* **27** (1983), 155–199.
[47] A. V. Molchanov, On definability of hypergraphs by their semigroups of homomorphisms, *Semigroup Forum* **62** (2001), 53–65.
[48] Gábor Nagy & Petr Vojtěchovský, *The LOOPS Package - a GAP package, Version 3.4.1*; 2018, `https://gap-packages.github.io/loops/`.
[49] P. M. Neumann, Primitive permutation groups and their section-regular partitions, *Michigan Math. J.*, **58** (2009), 309–322.
[50] B. M. Schein, Ordered sets, semilattices, distributive lattices and Boolean algebras with homomorphic endomorphism semigroups, *Fund. Math.* **68** (1970), 31–50.
[51] C. Schneider and A. C. Silva, Cliques and colorings in generalized Paley graphs and an approach to synchronization, *J. Algebra Appl.*, **14** (2015), no. 6, 13 pp.
[52] J. Schreier, Über Abbildungen einer abstrakten Menge Auf ihre Teilmengen, *Fund. Math.* **28** (1936), 261–264.
[53] J. Rhodes and B. Steinberg, The q-theory of finite semigroups. Springer Verlag. (2008).
[54] Shahzamanian, B. Steinberg, Simplicity of augmentation submodules for transformation monoids. *Algebr. Represent. Theory* **24** (2021), no. 4, 1029–1051.
[55] Smith, JDH, *Mal'cev varieties*, Lecture Notes in Mathematics, Vol. 554, 1976
[56] L. Soicher, *GRAPE: Graph algorithms using permutation groups*; version 4.2, `http://www.maths.qmul.ac.uk/~leonard/grape/`.
[57] M. R. Sorouhesh, On ideals of quasi-commutative semigroups. *Bull. Iranian Math. Soc.* **45** (2019), no. 2, 447–453.
[58] R. P. Sullivan, Automorphisms of transformation semigroups, *J. Austral. Math. Soc.* **20** (1975), 77–84.
[59] È. G. Šutov, Homomorphisms of the semigroup of all partial transformations, *Izv. Vysš. Učebn. Zaved. Matematika* **3** (1961), 177–184. (Russian)
[60] S.M. Ulam, "A Collection of Mathematical Problems," Interscience Publishers, New York, 1960.

Bibliografia

[1] Raymond M. Smullyan. First-Order Logic. Dover Publications, Inc. New York. 1995.

[2] Mordechai Ben-Ari. Mathematical Logic for Computer Science (3rd ed.). Springer-Verlag London. 2012.

[3] Ernesto Costa e Anabela Simões. Inteligência Artificial. Fundamentos e Aplicações (3 ed.). FCA. 2008.

[4] João P. Martins. Lógica e Raciocínio. Ensino da Ciência e da Tecnologia. IST Press. 2021.

[5] Melvin Fitting. First-Order Logic and Automated Theorem Proving (2nd ed.). Springer-Verlag New York. 1990.

[6] John Howie. Fundamentals of Semigroup Theory, London Mathematical Society Monographs volume 12. Oxford University Press New York. 1995.

[7] Keith Conrad. Conjugation in a group. Expository papers. https://kconrad.math.uconn.edu/blurbs/grouptheory/conjclass.pdf

[8] M. Saqib Nawaz, Moin Malik, Yi Li, Meng Sun and M. Ikram Ullah Lali. A Survey on Theorem Provers in Formal Methods. 2019. Pre-print. https://arxiv.org/abs/1912.03028

[9] John Harrison. A Short Survey of Automated Reasoning. Algebraic Biology 2007 Lecture Notes in Computer Science. volume 4545. Springer Berlin Heidelberg. 2007.

[10] Salvador Lucas. Proving Program Properties as First-Order Satisfiability. 28th International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR. 2018.

[11] Michael K. Kinyon. Proof simplification and automated theorem proving. Philosophical Transactions of The Royal Society. A Mathematical Physical and Engineering Sciences 377. March 2019.

[12] M. Saqib Nawaz, Moin Malik, Yi Li, Meng Sun and M. Ikram Ullah Lali. A Survey on Theorem Provers in Formal Methods. Pre-print. 2019. https://arxiv.org/abs/1912.03028

[13] Jorge Raminhos. Varieties of Semigroups. 2019. https://sgv.pythonanywhere.com

[14] João Araújo, João Pedro Araújo, Peter J. Cameron, Edmond W. H. Lee and Jorge Raminhos. A Survey on Varieties Generated by Small Semigroups and a Companion Website, 2019, Preprint https://arxiv.org/abs/1911.05817

[15] Laura Kovács and Andrei Voronkov, First-Order Theorem Proving and Vampire. In: Sharygina N., Veith H. (eds) Computer Aided Verification. CAV 2013. Lecture Notes in Computer Science, vol 8044. Springer, Berlin, Heidelberg. 2013.

[16] McCune, W. Mace4 Reference Manual and Guide. Tech. Memo ANL/MCS-TM-264, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, August 2003.

[17] João Araújo, Fernando Maia Ferreira and Michael Kinyon. Varieties of Lazy Magmas Characterized by Forbidden Substructure Theorems. Pre-print. 2022 https://arxiv.org/abs/2205.14343

[18] João Araújo, Choiwah Chow and Mikoláš Janota. Boosting Isomorphic Model Filtering with Invariants. Pre-print. 2022 https://arxiv.org/abs/2201.10516

[19] João Araújo, Rui Barradas Pereira, Wolfram Bentz, Choiwah Chow, João Ramires, Luis Sequeira and Carlos Sousa. CREAM: a Package to Compute [Auto, Endo, Iso, Mono, Epi]-morphisms, Congruences, Divisors and More for Algebras of Type $(2^n, 1^n)$. Pre-print. 2022 https://arxiv.org/abs/2202.00613

[20] Peter Jipsen. Mathematical Structures. https://math.chapman.edu/~jipsen/structures/doku.php

[21] Peter Höfner. Algebraic Reasoning with Prover9. 2007. http://www.kleenealgebra.de/kleene_db/lemmas/all.php

[22] R. Padmanabhan and Yang Zhang. Commutativity Theorems in Groups with Power-like Maps. Journal of Formalized Reasoning Vol.12, No.1, pp. 1-10, 2019. https://jfr.unibo.it/article/view/8751/8968

[23] Rob Arthan and Paulo Oliva. Studying Algebraic Structures Using Prover9 and Mace4. Proof Technology in Mathematics Research and Teaching . Mathematics Education in the Digital Era, vol 14. Springer, Cham. 2019. https://arxiv.org/abs/1908.06479

[24] Michael K. Kinyon, Kenneth Kunen and J. D. Phillips. Every Diassociative A-Loop is Moufang. Proceedings of the American Mathematical Society. Vol. 130, No. 3, pp. 619-624, 2002. https://arxiv.org/abs/math/0007195

[25] Tomávs Kepka, Michael K. Kinyon and J. D. Phillips. The structure of F-quasigroups. Journal of Algebra Volume 317, Issue 2, pp 435-461, 2007. https://arxiv.org/abs/math/0510298

[26] Scerbacov Victor and Izbas Vladimir. A quasigroup with any of Moufang identities is a loop. Buletinul Academiei de Stiinte a Republicii Moldova. Matematica BASM n.2 (27), 1998.

[27] Tadeusz Litaka, Szabolcs Mikulásb and Jan Hidders. Relational Lattices: From Databases to Universal Algebra. Journal of Logical and Algebraic Methods in Programming. Volume 85, Issue 4, pp 540-573, 2016.

[28] C. A. R. Tony Hoare, Bernhard Möller, Georg Struth and Ian Wehrman. Concurrent Kleene Algebra. Lecture Notes in Computer Science, vol 5710. Springer, Berlin, Heidelberg, 2009. https://doi.org/10.1007/978-3-642-04081-8_27