# UMA LINGUAGEM COMPUTACIONAL DE REESCRITA DE EXPRESSÕES MATEMÁTICAS POR VIA AXIOMÁTICA

por

Jaime Augusto Alves dos Remédios

Dissertação
apresentada na Universidade Aberta
para a obtenção do grau de
Doutor em Informática

Lisboa, Setembro 2007

# UMA LINGUAGEM COMPUTACIONAL PARA A REESCRITA DE EXPRESSÕES MATEMÁTICAS POR VIA AXIOMÁTICA

Sob a orientação do
Professor Doutor Alexandre Cerveira
Professor Catedrático do
Departamento de Ciências Exactas e Tecnológicas da
Universidade Aberta

e do

Prof. Doutor Nuno Cavalheiro Marques
Professor Auxiliar do
Departamento de Informática da
Faculdade de Ciências e Tecnologia da
Universidade Nova de Lisboa

# AGRADECIMENTOS

Os meus sinceros agradecimentos vão para os meus orientadores de tese, Professor Doutor Alexandre Cerveira e Prof. Doutor Nuno Cavalheiro Marques, que aceitaram este desafio e que sempre acreditaram que eu seria capaz de levar este projecto a bom porto. Um especial agradecimento ao Doutor Nuno Marques por ter sugerido, logo na primeira reunião, a utilização do Prolog. As inúmeras reuniões que tivemos foram sempre esclarecedoras e bastante motivantes. Gostaria também de agradecer ao Professor Doutor José Pereira Neto e ao Prof. Doutor Fausto Amaro pelo apoio que sempre me deram quando ainda eram os Administradores da Universidade Internacional, e que sempre me motivaram a continuar, mesmo quando deixaram de o ser, e o apoio me foi retirado pela nova Administração. Gostava também de agradecer aos poucos amigos que me apoiaram até ao fim, em especial, ao Prof. Doutor Armando de Oliveira, que sempre teve uma palavra de conforto e de amizade, nos momentos de maior dificuldade, quando tudo parecia desmoronar à minha volta. Os meus últimos dois anos de doutoramento foram penosos e bastante difíceis. Gostaria de agradecer, em especial, aos meus dois filhos, Pedro e Andrew, que estoicamente, se mantiveram ao meu lado, e que me deram apoio moral, quando eu mais necessitava, e que me ajudaram a ultrapassar as enormes dificuldades financeiras por que tive que passar no fim deste projecto. A eles dedico esta tese e a minha eterna gratidão.

# ABSTRACT

The simplification of algebraic expressions can be accomplished at the most atomic or elementary step. This thesis introduces a new method of simplifying algebraic expressions, using 1st order unification and term rewriting methods. Algebraic expressions are viewed as triangular structures, of finite depth, and are rewritten, axiomatically, into a finite set of equivalent forms that leads to their normalization. Rewriting is done by matching these expressions with the left sides of mathematical formulas with the same triangular structure. The simplification method, defended by this thesis, produces complete symbolic solutions as opposed to straight, either numeric or symbolic, answers. A symbolic solution is a solution that reflects the mathematical reasoning behind the resolution process. Research in symbolic computing has already produced many outstanding results in the area of term unification and rewriting. Our simplification method is built on this research. By viewing these expressions as triangular structures, it was possible to develop very efficient ways of matching algebraic expressions and mathematical rules by classes of equivalence. The method that is defended in this thesis is based on standard unification and logic composition of mathematical expressions, using 1st order functional constructors and oriented conditional term rewriting. The method was implemented on Prolog and uses DCG as a natural language processor for mathematical expressions. The majority of computer algebra systems (CAS), such as Maple™ and Mathematica™, use higher order logics and do not produce symbolic solutions at the elementary level. The program MathXPert™ and the derivation system developed for the EPGY program of the University of Stanford produce these kinds of solutions but use also higher order logics. Our contribution was the development of a triangular structure as a logic structure for symbolic computation. It was thanks to this kind of structure that it was possible to develop complete symbolic solutions to algebraic simplification problems using only 1st

order logic.

# RESUMO

A simplificação de expressões algébricas pode ser feita ao nível mais atómico ou elementar. Esta tese introduz um novo método de simplificar expressões algébricas, utilizando métodos de unificação e reescrita de termos de 1ª ordem. As expressões algébricas são vistas como estruturas triangulares, de profundidade finita, e são reescritas, axiomaticamente, num número finito de formas equivalentes que conduzem à sua normalização. A reescrita é feita comparando essas expressões com o lado esquerdo de fórmulas matemáticas com a mesma estrutura triangular. O método de simplificação, defendido nesta tese, produz soluções simbólicas completas, e não apenas respostas directas, numéricas ou simbólicas. Uma solução simbólica é uma solução que reflecte o raciocínio matemático que está por trás do processo de resolução simbólica. A pesquisa em computação simbólica tem produzido resultados bastante significativos na área da unificação e reescrita de termos. O nosso método de simplificação foi desenvolvido com base nos resultados obtidos dessa pesquisa. Com o conceito de estrutura triangular, foi possível desenvolver formas bastante eficientes de comparar expressões algébricas e regras matemáticas por classes de equivalência. O método de simplificação, que é defendido nesta tese, é baseado no método convencional de unificação e na composição lógica de expressões matemáticas, utilizando construtores funcionais e regras de reescrita condicional de 1ª ordem. O método foi implementado em Prolog e utiliza o DCG como processador da linguagem corrente de expressões matemáticas. A maioria dos sistemas de computação algébrica (CAS), como por exemplo o Maple™ e o Mathematica™, utilizam lógicas de ordem superior e não produzem soluções simbólicas ao nível elementar. O programa MathXPert™ e o sistema de derivação desenvolvido para o programa EPGY da Universidade de Stanford produzem esse tipo de solução mas utilizando também lógicas de ordem superior. A nossa contribuição foi o desenvolvimento da

estrutura triangular como uma estrutura lógica para a computação simbólica de expressões matemáticas. Foi graças a este tipo de estrutura que foi possível desenvolver soluções simbólicas para problemas de simplificação algébrica, utilizando apenas lógicas de 1ª ordem.

# ÍNDICE

# Índice

# LISTA DE DEFINIÇÕES

# Lista de Definições

# LISTA DE FIGURAS

# Lista de Figuras

# GLOSSÁRIO INGLÊS - PORTUGUÊS

| | |
|---|---|
| Automated deduction | Dedução automatizada |
| Atomic sentence | Frase atómica |
| Backtracking | Retrocesso |
| Binding | Ligação |
| Black box | Caixa preta |
| Breadth-first search | Busca em extensão |
| Closed world assumption | Hipótese do mundo fechado |
| Definite clause | Cláusula definitiva |
| Depth-first search | Busca em profundidade |
| Falsity | Falsidade |
| Glass box | Caixa de vidro |
| Goal | Objectivo |
| Goal clause | Cláusula objectiva |
| Goal resolution | Resolução do objectivo |
| Ground term | Termo fechado |
| Innermost term | Termo mais interior |
| Lazy strategy | Estratégia retardada |
| Least Herbrand model | Menor modelo de Herbrand |
| Like terms | Termos afins |
| Linear resolution | Resolução linear |
| Negation by failure | Negação por insucesso |
| Outermost term | Termo mais exterior |
| Parsing | Análise sintáctica |
| Reducible expression (redex) | Expressão redutível |
| Sentence | Frase |
| Strict (eager) strategy | Estratégia estrita |

| | |
|---|---|
| String | Cadeia de caracteres |
| Term rewrite | Reescrita de termos |
| Top-down | Descendente |
| Tree structure | Estrutura arbórea |
| White box | Caixa branca |

# GLOSSÁRIO PORTUGUÊS - INGLÊS

| | |
|---|---|
| Análise sintáctica | Parsing |
| Busca em extensão | Breadth-first search |
| Busca em profundidade | Depth-first search |
| Cadeia de caracteres | String |
| Caixa de vidro | Glass box |
| Caixa branca | White box |
| Caixa preta | Black box |
| Cláusula definitiva | Definite clause |
| Cláusula objectiva | Goal clause |
| Dedução automatizada | Automated deduction |
| Descendente | Top-down |
| Estratégia estrita | Strict (eager) strategy |
| Estratégia retardada | Lazy strategy |
| Estrutura arbórea | Tree structure |
| Expressão redutível | Reducible expression (redex) |
| Frase | Sentence |
| Frase atómica | Atomic sentence |
| Hipótese do mundo fechado | Closed world assumption |
| Ligação | Binding |
| Menor modelo de Herbrand | Least Herbrand model |
| Negação por insucesso | Negation by failure |
| Objectivo | Goal |
| Reescrita de termos | Term rewrite |
| Resolução do objectivo | Goal resolution |
| Resolução linear | Linear resolution |
| Retrocesso | Backtracking |

| | |
|---|---|
| Termos afins | Like terms |
| Termo fechado | Ground term |
| Termo mais exterior | Outermost term |
| Termo mais interior | Innermost term |
| Verdade | Truth |

# Capítulo 1 Introdução

## MOTIVAÇÃO E OS OBJECTIVOS DA TESE

## 1.1 Os Objectivos da Tese

As palavras proferidas pelo Professor Allen[1] [ALL88],

> **"We think in terms of words. Yet today anyone who identifies lexical reasoning, based on gradually formalized natural language, as the key to the learning of mathematics, is filing a minority report."[2]**

ilustram bem a importância que as palavras têm no desenvolvimento da capacidade de interpretação e de raciocínio e, portanto, na aprendizagem da Matemática como uma linguagem natural. Mas para isso é necessário que tanto a linguagem como os termos que nela são empregues – as expressões matemáticas – sejam, tanto quanto possível, inequívocas. A Matemática não é apenas uma linguagem de expressões. É também uma linguagem que emprega a lógica e o raciocínio na manipulação dessas expressões.

O que nos motivou para este trabalho de investigação foi precisamente a importância que a simplificação algébrica desempenha no ensino e aprendizagem da Matemática. As expressões que estudámos para esta tese são do tipo que é

[1] Palavras extraídas do discurso proferido por Frank B. Allen, Professor Emérito de Matemática da Elmhurst College, no Encontro Anual da NCTM em Chicago, Abril 1988. O Professor Allen foi um dos Presidentes da NCTM e era nessa altura o Conselheiro Nacional do "*Mathematically Correct*".

[2] "*Nós pensamos em termos de palavras. No entanto, quem hoje identifique a capacidade de interpretação e raciocínio, com base numa linguagem natural, gradualmente, formalizada, como sendo a chave para a aprendizagem da Matemática, faria, certamente, parte de uma minoria.*" (Tradução)

normalmente utilizado no ensino secundário. O número de operadores matemáticos incluídos neste estudo é restrito assim como a classe de expressões que resultam da composição lógica desses operadores como estruturas binárias. Neste estudo não foram incluídos os polinómios.

Como sabemos, a Matemática não é uma linguagem natural, no sentido em que o são, por exemplo, linguagens como o Português [MAT96]. De facto, na sua forma *escrita*, a Matemática é uma linguagem que poderíamos classificar como, essencialmente, gráfica e, predominantemente, abstracta. Na sua forma *falada*, porém, a Matemática, em pouco ou nada, se distingue dessas outras linguagens. Frases, como "*calcular a primeira derivada do logaritmo natural do quadrado de x*", ou "*simplificar o logaritmo decimal do cubo de 10*", são frases que classificaríamos facilmente como naturais. Nessas frases, as expressões matemáticas estão escritas em linguagem corrente.

Uma grande parte do ensino da Matemática, ao nível do secundário, tem que ver precisamente com a capacidade de interpretação e manipulação de expressões matemáticas. *Fazer* Matemática é, em grande parte, isso – interpretar e manipular expressões. De facto, as expressões matemáticas podem ser lidas de uma forma que poderíamos classificar de operacional. Por exemplo, a expressão $\log_e x^2$ pode ser lida operacionalmente como "*log e pwr 2 x*". Com esse tipo de leitura é possível obter interpretações inequívocas dessas expressões. Uma leitura operacional é uma forma de leitura baseada na representação lógica de expressões matemáticas como estruturas arbóreas binárias. Um tipo de estrutura que designamos por estrutura triangular (Secção 2.4). Por exemplo, a expressão $\log_e x^2$ pode ser representada por uma expressão lógica, do tipo $\log(e, \text{pwr}(2, x))$. Numa expressão lógica, os operadores matemáticos estão representados de uma forma explícita e prefixa, como operadores binários. Essa forma de representação é baseada na notação polaca.

Com base nesse tipo de representação, é possível simplificar, de forma

simbólica, a classe de expressões matemáticas, cuja caracterização pode ser obtida por leitura operacional da sua componente principal. Por exemplo, a expressão $\log_e 4^2$, pode ser interpretado operacionalmente como o "*logaritmo de uma potência*". Já o mesmo não sucede com polinómios e outras expressões cujo significado operacional não se pode obter dessa forma. Por exemplo, o polinómio $x^2+2x+4$ não pode ser interpretado operacionalmente como a "*soma de uma soma*". O mesmo se poderia dizer de expressões como $\sqrt{\sqrt{3}+\sqrt{2}}$ ou $4^{4^x}$. O significado operacional de uma expressão matemática é simplesmente, o resultado de uma leitura simbólica da sua representação lógica. Essa leitura envolve apenas a componente principal. Por exemplo, a expressão $\log_e \sqrt{x^3}$ é interpretada como o "*logaritmo de uma raiz*". A leitura simbólica de uma representação lógica é simplesmente uma leitura sequencial dos seus símbolos, da esquerda para a direita, excluindo os parênteses e vírgulas. As componentes de uma expressão são interpretadas e simplificadas separadamente. Essa forma operacional de interpretar expressões matemáticas permite que as expressões possam ser simplificadas de uma forma simbólica que designamos por resolução simbólica. Através de uma leitura operacional podem ser facilmente identificados, de uma forma explícita ou implícita, os contextos em que estão inseridas as expressões.

Uma resolução simbólica é, simplesmente, uma sequência ordenada de transformações lógicas efectuadas, por reescrita axiomática, sobre uma expressão, e que visam essencialmente a obtenção da forma mais simples dessa expressão. Esse tipo de reescrita está intimamente ligado ao significado operacional das expressões. Uma reescrita axiomática não é mais do que uma transformação lógica obtida por via axiomática. A base axiomática de uma resolução simbólica é simplesmente um conjunto finito de regras de reescrita (axiomas e teoremas), condicionais, organizado por classes. A classe de uma expressão é determinada, única e exclusivamente, pela sua assinatura, ou seja, pela estrutura lógica da sua

componente principal. De facto, podemos dizer que a classe de uma expressão é determinada pela forma inequívoca como a sua componente principal é lida. Por exemplo, expressões como $\log_e(x+1)^2$ e $\log_e x^2$, possuem a mesma assinatura, *log_pwr*. Ambas são expressões que poderíamos caracterizar como "*logaritmos de potências*", ou seja, expressões com uma estrutura lógica do tipo log(_, pwr(_, _)) [3].

Tal como em qualquer linguagem, também na Matemática, os símbolos fazem-nos agir e reagir de acordo com o seu significado. Em Matemática, porém, os padrões simbólicos representam expressões com significados operacionais bem definidos. A reescrita axiomática é, precisamente, a transformação lógica desses padrões, por reescrita. Por exemplo, a expressão $\log_e(x+1)^2$ pode ser reescrita como $2\log_e(x+1)$, aplicando-lhe uma regra de reescrita da sua classe. Naturalmente, a reescrita de uma expressão depende da regra que lhe é aplicada e da parte da estrutura em que essa regra é aplicada, ou seja, da estratégia que é escolhida para a simplificar. É, precisamente, através de uma leitura operacional desses padrões, que essas estratégias são determinadas.

Verificou-se que, relativamente ao conjunto de axiomas em estudo, uma reescrita axiomática depende apenas da caracterização da expressão relativamente a essa base. Caracterização que é feita com base naquilo que designamos por assinatura (Secção 2.4.1). O resultado de uma reescrita é uma expressão equivalente que tanto pode ser irredutível (normal) como irredutível (redutora ou potencialmente redutora). Por exemplo, a expressão $\log_e x^2$ pode ser transformada numa outra, que lhe é equivalente, $2\log_e x$, mas irredutível. Pensamos também que muitas outras expressões matemáticas, como por exemplo, equações algébricas ($2x+1=3x-2$) ou inequações ($3x+1>4$), poderiam ser tratadas também por reescrita axiomática.

[3] O símbolo " _ " é utilizado, aqui, para representar argumentos, tanto atómicos como funcionais (ver Secção 2.2.1).

Pretendemos, com esta tese, demonstrar que

***Hipótese I:***

> *É possível resolver, de forma simbólica e por reescrita axiomática, qualquer expressão matemática que possa ser completamente caracterizada pela estrutura triangular da sua componente principal e que seja simplificável.*

A caracterização é feita através da assinatura. Para resolver expressões matemáticas, de uma forma simbólica e por reescrita axiomática, é necessário aplicar um certo número de regras de reescrita e estratégias de resolução. Uma expressão é simplificável se existirem regras de reescrita com a mesma assinatura que reduzam o seu grau de complexidade, ou aumentem o seu poder redutor.

A resolução simbólica de expressões matemáticas visa essencialmente a simplificação dessas expressões, através da reescrita axiomática de uma, ou mais, formas equivalentes dessas expressões. Por exemplo, para resolver uma expressão, como $\log_e 4^2$, de uma forma simbólica e por reescrita axiomática, é necessário aplicar um certo número de regras de reescrita. A aplicação dessas regras é determinada pela classe de reescrita dessa expressão. A classe de reescrita de uma expressão define o conjunto de assinaturas que podem intervir na resolução dessa expressão. Resolver uma expressão é obter uma solução simbólica dessa expressão, ou seja, obter uma sequência lógica de formas equivalentes dessa expressão, que conduzam à forma normal dessa expressão. No caso da expressão $\log_e 4^2$, uma possível solução seria uma sequência do tipo[4]

$$\log_e 4^2 \rightarrow_\alpha 2\log_e 4 \downarrow^1 \rightarrow_\gamma 2\log_e 2^2 \downarrow^1 \rightarrow_\alpha 2(2\log_e 2) \rightarrow_\alpha$$

[4] As letras gregas, nessa sequência, representam o tipo de reescrita aplicada nessas transformações. As setas, $\downarrow$ e $\rightarrow$, representam o tipo de estratégia aplicada. As setas denotam também uma relação de equivalência entre os vários elementos dessa sequência.

$$\rightarrow_{\alpha} 2(2)\log_e 2 \downarrow^{1} \rightarrow_{\kappa} 4\log_e 2$$

Cada solução é o resultado da aplicação de um certo número de estratégias. Todas as formas equivalentes de uma expressão, obtidas por reescrita axiomática, possuem significados operacionais bem definidos, isto é, podem ser caracterizadas pelas estruturas triangulares das suas respectivas componentes principais.

As várias formas de resolver, simbolicamente, uma expressão matemática, diferem apenas no tipo de raciocínio que é aplicado na resolução. Solow [SOL82] compara esse tipo de raciocínio com aquele que é, normalmente, utilizado na resolução de problemas de labirintos. Segundo Solow, a resolução desse tipo de problemas pode ser feito, por avanços e recuos, aplicando estratégias de resolução que utilizem métodos de busca em profundidade (*depth-first search*), métodos de busca em extensão (*breadth-first search*), ou métodos que sejam uma combinação de ambos.

Verificámos também que a aplicação de estratégias depende do tipo de operadores matemáticos presentes nessas expressões. Por exemplo, para se resolver uma expressão, como $\log_e \sqrt{4^3}$, pode-se aplicar uma estratégia de simplificação, para logaritmos, e resolver a componente $\sqrt{4^3}$. Um dos factores que influencia a escolha de uma estratégia é, precisamente, o tipo de estrutura triangular, que está subjacente a essas expressões. Uma estrutura triangular é uma estrutura arbórea binária, com uma assinatura bem definida.

Um dos principais objectivos desta tese foi, precisamente, o desenvolvimento de uma estrutura computável para a resolução simbólica de expressões matemáticas por reescrita axiomática. O resultado foi o desenvolvimento da estrutura triangular como forma de representação lógica de expressões matemáticas. O que nos motivou foi o facto de ser este o tipo de

estrutura exibido pelas regras matemáticas (axiomas e teoremas) que servem de base axiomática à resolução simbólica.

A resolução simbólica de expressões visa, essencialmente, a simplificação dessas expressões ou seja, a aplicação de um certo número de regras de reescrita que reduzam, ou contribuam para a redução do grau de complexidade dessas estruturas.

Pretendemos, com esta tese, demonstrar também que

***Hipótese II:***

> *A resolução simbólica, por reescrita axiomática, de expressões matemáticas com estruturas triangulares, é convergente se a base axiomática for completa.*

Com o desenvolvimento da estrutura triangular, foi possível definir uma forma de leitura, totalmente computacional, baseada na representação lógica de expressões matemáticas, como estruturas arbóreas binárias, com uma assinatura bem definida. Por exemplo, uma expressão, como $\log_e \sqrt{4^3}$, com uma estrutura triangular do tipo $\log(\_, \text{root}(\_, \text{pwr}(\_, \_)))$ pode ser lida, operacionalmente, como uma expressão com uma assinatura *log_root* e duas componentes, *root_pwr* e *pwr*. As componentes são resolvidas separadamente como estruturas triangulares e o resultado composto de uma forma lógica por reescrita também. A convergência de uma resolução simbólica determina a unicidade do seu resultado. O resultado de uma resolução simbólica é a forma mais simples dessa expressão. O conceito de simplicidade, nesta tese, é determinado apenas, comparativamente, com outras formas equivalentes da mesma expressão. O grau de complexidade de uma expressão não é um valor intrínseco da expressão. O grau de complexidade de uma expressão é determinado, comparativamente, pelo seu número de componentes e a

estrutura triangular das suas componentes

Para o desenvolvimento de uma linguagem de resolução simbólica baseada na estrutura triangular, como forma de representação lógica de expressões matemáticas, e na reescrita axiomática como método de transformação lógica de expressões, optámos por uma linguagem de programação lógica de 1ª ordem, como o Prolog. O Prolog reúne todos os requisitos necessários para o desenvolvimento de uma linguagem deste tipo . A base axiomática está representada por um conjunto de cláusulas definitivas ("*definite clauses*"), devidamente classificadas por tipo de assinatura. O acesso à base é feito de forma sequencial e por unificação (Secção 3.3.2). A reescrita axiomática é simplesmente o resultado da unificação de uma expressão lógica com uma regra de reescrita com a mesma assinatura. A assinatura de uma expressão define a sua classe. Por exemplo, a relação fundamental entre logaritmos e potências está representada por um conjunto de cláusulas, onde o *lado esquerdo* da relação é representado por um termo do tipo $\log(Arg_1, \text{pwr}(Arg_{21}, Arg_{22}))$. Cláusulas desse tipo unificam apenas com expressões cujas assinaturas são do tipo *log_pwr* (logaritmos de potências).

Beeson e Ravaglia desenvolveram sistemas semelhantes, mas baseados em lógicas de ordem superior. O sistema desenvolvido por Beeson, Mathpert, foi totalmente desenvolvido em C, e mais tarde comercializado com o nome de MathXpert™ (2000). A sua comercialização tornou, de certo modo, difícil o acesso a certos aspectos técnicos do sistema, como por exemplo o tipo de estruturas e métodos utilizados na representação e transformação de expressões. Ravaglia, por outro lado, desenvolveu um sistema de derivação com base no sistema Maple™. O Maple™ é utilizado, aqui, como motor inferencial.

A principal contribuição desta tese foi o desenvolvimento da noção de estrutura triangular, como forma de representação lógica de expressões matemáticas. Com base nesse tipo de estrutura, foi possível desenvolver um método

de resolução simbólica, bastante eficiente, computacionalmente, baseado na reescrita axiomática de expressões matemáticas. Como aplicação prática, foi desenvolvido um programa, em Prolog, capaz de resolver, simbolicamente, e passo a passo, expressões matemáticas escritas em linguagem corrente (natural) ou abreviada (quasi-natural).

Os métodos que desenvolvemos utilizam o método de resolução SLDNF (*Linear Resolution for Definite Clauses with Selection Function and Negation by Failure*) do Prolog, como motor inferencial. A reescrita de expressões é feita por composição lógica de construtores-$\lambda$. A reescrita axiomática é feita por unificação (Secção 3.3.2) dessas expressões com regras de reescrita, condicionais, organizadas hierarquicamente por classes e ordenadas por ordem decrescente do poder redutor. A aplicação de estratégias é controlada pelo princípio de exclusão e *backtracking*.

## 1.2 Organização do Documento

No Capítulo 2 é abordado o objecto deste estudo, ou seja, a resolução simbólica de expressões matemáticas por via axiomática. São tratadas, aí, questões sobre a representação lógica de expressões matemáticas, o processo de transformação lógica como uma forma de reescrita de expressões por via axiomática, e a resolução simbólica como uma sequência de transformações lógicas que conduzem à simplificação de expressões. É aí que é discutida também a estrutura triangular como uma forma computacional, bastante eficiente, de representar expressões matemáticas.

No Capítulo 1 é abordado o trabalho relacionado e a base teórica em que o nosso trabalho se baseou. Nesse capítulo é discutida a representação lógica de expressões matemáticas utilizando uma linguagem lógica de 1ª ordem, os métodos de resolução e unificação utilizados por uma linguagem de programação lógica

como o Prolog, a análise de expressões utilizando gramáticas DCG e a reescrita condicional de termos.

No Capítulo 4 é discutido o nosso trabalho de investigação, ou seja, a linguagem desenvolvida, em Prolog, para a resolução simbólica de expressões matemáticas, por via axiomática. Nesse capítulo é abordada a formação de expressões por composição lógica e construção-$\lambda$, e a transformação lógica de expressões matemáticas, por reescrita axiomática. É aí que são discutidos os vários tipos de reescrita, a memória colectiva de uma resolução e o princípio de exclusão como uma forma de garantir a convergência do sistema de reescrita.

No Capítulo 1 é discutido o Assistente de Matemática que desenvolvemos com base nessa linguagem. O Assistente utiliza uma interface gráfica, desenvolvida em Java, para a escrita de expressões matemáticas, em linguagem natural ou quasi-natural, e para a resolução simbólica dessas expressões.

Finalmente, no Capítulo 6, são discutidas as conclusões a que chegámos e que têm a ver, essencialmente, com as duas hipóteses que nos propusemos defender nesta tese. Nesse capítulo são abordadas também algumas das ideias sobre o nosso trabalho futuro.

O Apêndice A apresenta a implementação Prolog de algumas das estratégias e regras que são utilizadas numa resolução simbólica de expressões matemáticas.

O Apêndice B lista os axiomas e teoremas em que a reescrita axiomática se baseou.

O Apêndice C apresenta alguns dos exemplos resolvidos, nesta tese, por reescrita axiomática.

# Capítulo 2 O Objecto de Estudo

### A Resolução Simbólica de Expressões Matemáticas por Via Axiomática

## 2.1 Introdução

A Matemática é uma linguagem com uma gramática e vocabulário próprios. Como tal, está sujeita, como todas as outras, a regras, bem definidas, de composição e sintaxe. A linguagem corrente da Matemática utiliza operadores binários. Assim, quando se pede a alguém, para "*simplificar o logaritmo natural do quadrado de 4*", está-se, de facto, a *falar* Matemática. Frases, como essa, induzem-nos a raciocínios, mais ou menos, elaborados do que se deve fazer com expressões, como o "*logaritmo natural do quadrado de 4*". São frases que colocam questões operacionais sobre expressões matemáticas.

*Resolver expressões matemáticas, de uma forma simbólica,* é obter soluções simbólicas que reflictam, de uma forma detalhada e completa, o tipo de raciocínio que foi empregue nesse tipo de resolução. Quando se pede a um aluno do ensino secundário para "*simplificar o logaritmo natural do quadrado de 4*", está-se, de facto, a pedir que ele/a faça algo que tem a ver com aquilo que designamos por significado operacional dessa expressão. Por outras palavras, está-se a pedir ao aluno que responda à questão "*Qual é a forma mais simples da expressão* $\log_e 4^2$ *?*". Ou seja, que transforme essa expressão numa forma equivalente mas menos complicada. Neste caso, a resposta seria $4\log_e 2$. Mas, nem todas as questões, em Matemática, são tão simples como essa. Questões, como "*calcular a segunda*

*derivada do logaritmo natural da raiz cúbica do quadrado de x*", exigem uma exposição mais detalhada do tipo de raciocínio[5] que deve ser utilizado nessa resolução. Mas será que o aluno é capaz de o fazer? Ele sabe que vai ter de aplicar um número de axiomas e/ou teoremas para o fazer. Mas quais e como?

O estudo apresentado nesta tese foca o tipo de raciocínio que é normalmente utilizado na simplificação de expressões algébricas. O tratamento de expressões algébricas requer uma base axiomática relativamente completa. Nesta tese, vamos tratar dos axiomas mais comuns, possibilitando assim um tratamento de expressões matemáticas relativamente completa e, para tal, restringirmo-nos aos que são tratados no ensino secundário.

Para responder a esse tipo de questões, é necessário definir primeiro o que entendemos por expressão matemática e quais as formas de a representar. O que é então uma *expressão matemática*? A resposta a esta questão leva-nos, forçosamente, a considerações de representatividade e, portanto, a aspectos de natureza linguística.

Uma expressão matemática pode ser representada de várias formas.

- **Gráfica** – Nesta forma de representação são utilizadas formas especiais de escrita (como por exemplo, índices e expoentes), para representar certos operadores matemáticos de uma forma implícita. Graficamente, os operadores matemáticos podem ser representados de uma forma explícita ou implícita. Esta forma de representação é a forma que é, normalmente, conhecida por notação matemática. Uma expressão, como $\log_e 4^2$, estaria representada nessa forma;

- **Linear** – Nesta forma de representação são utilizados símbolos especiais (como por exemplo, o símbolo "^"), para representar certos

[5] A exposição de raciocínios matemáticos é também utilizada noutras áreas da Matemática como, por exemplo, na demonstração de teoremas, ou a resolução de "*word problems*". Nesta tese, estamos apenas interessados no tipo de raciocínio dedutivo que é utilizado na simplificação de expressões.

operadores matemáticos, como por exemplo potências, de uma forma explícita, mas linear. Esta é a forma normalmente utilizada na escrita computadorizada de expressões matemáticas. Nessa forma de escrita, os operadores matemáticos estão representados de uma forma infixa, ou prefixa. Por exemplo, uma expressão, como $\log(e, 4 \wedge 2)$, seria a forma linear de se representar a expressão gráfica, $\log_e 4^2$;

- **Lógica** – Nesta forma de representação são utilizados símbolos lexicográficos (como por exemplo, "*log*" ou "*pwr*"), para representar os vários operadores matemáticos, de uma forma explícita mas também estritamente linear. Nessa forma de escrita, os operadores matemáticos estão representados de uma forma prefixa e binária. Esta foi, precisamente, a notação que escolhemos para representar expressões matemáticas. Por exemplo, uma expressão, como $\log(e, \text{pwr}(2,4))$, seria a forma lógica de se representar a expressão gráfica $\log_e 4^2$;

- **Quasi-natural** – Nesta forma de representação, semelhante à lógica, os elementos estruturais (como por exemplo, parênteses e vírgulas) são substituídos por espaços. Esta foi a notação que escolhemos para escrever (*input*) expressões matemáticas como árvores binárias. Por exemplo, uma expressão, como "*log e pwr 2 4*", seria a forma quasi-natural de se representar a expressão gráfica $\log_e 4^2$;

- **Natural** – Nesta forma de representação, semelhante à quasi-natural, a expressão é escrita em linguagem corrente. Por exemplo, uma expressão, como "*logaritmo natural do quadrado de 4*", seria a forma natural de representar a expressão gráfica $\log_e 4^2$.

Como veremos a seguir, a representação lógica é a forma de representação que melhor se adequa a uma forma de leitura que designamos por operacional ou computacional. Nessa forma, a expressão matemática é designada por expressão

lógica. Numa expressão lógica, os operadores matemáticos estão representados por formas prefixas e lineares. Veremos, na Secção 2.3.2, que as formas lineares com operadores infixos podem introduzir ambiguidades na leitura simbólica de expressões matemáticas.

## 2.2 A Representação Lógica de Expressões

Como sabemos, as expressões matemáticas possuem uma forma de leitura e de escrita, diferente daquela que é, normalmente, utilizada pela grande maioria das expressões em Português. Em Português, as expressões são lidas e escritas de uma forma que poderíamos classificar como linear e sequencial.

A forma de escrita que, aparentemente, alguns operadores matemáticos apresentam, como sendo linear e sequencial, é apenas ilusória. A soma e a subtracção são dois desses operadores. Essa aparente ilusão parece advir apenas do facto desses operadores serem, normalmente, interpretados como "*mais*" e "*menos*", respectivamente. A ilusão não existiria, se esses operadores fossem interpretados como "*soma de*" ou "*diferença entre*". Essa é aliás a forma como esses operadores são escritos em linguagem corrente. Na sua grande maioria, os operadores matemáticos possuem formas gráficas de escrita, que pouco ou nada têm a ver com a escrita, em linguagem corrente.

Como se sabe, quando se estuda uma linguagem, deve-se ter em linha de conta dois aspectos fundamentais:

- A forma como as expressões devem ser escritas (o aspecto sintáctico);
- A forma como as expressões devem ser lidas ou interpretadas (o aspecto semântico).

No caso de uma linguagem, como a apresentada nesta tese, que lida essencialmente com a resolução simbólica de expressões matemáticas, o aspecto sintáctico está ligado à forma como essas expressões devem ser introduzidas e representadas no computador. O aspecto semântico, por outro lado, está ligado ao significado operacional dessas expressões, ou seja, à forma como essas expressões devem ser resolvidas simbolicamente. A forma correcta e inequívoca de se ler uma expressão matemática é através de uma leitura operacional. Como veremos, uma leitura operacional confere à expressão um significado que poderíamos designar por operacional ou computacional.

Quando se *escreve*, por exemplo, a soma de dois números, como $2+3$, está-se, de facto, a *escrever* algo que tem a ver com o conceito, ou operação soma. É, precisamente, o significado daquilo que se escreve, que realmente importa. A forma como se escreve é meramente sintáctica. Quanto mais próximos estiverem esses dois conceitos, tanto melhor será o resultado obtido. Um bom livro é aquele que está escrito numa forma que todos entendem. Talvez seja essa a razão porque a Matemática é tão mal compreendida!

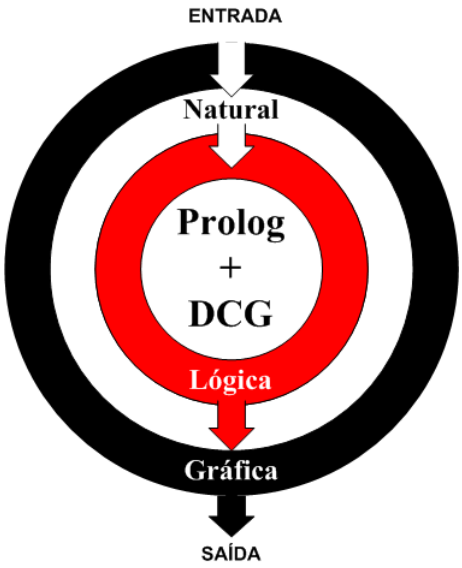



**Figura 2.2.1** *Estrutura da linguagem*

A linguagem apresentada nesta tese utiliza três níveis de representação (gráfico, natural ou quasi-natural, e lógico). Como está ilustrado na Figura 2.2.1, cada nível possui o seu próprio suporte linguístico. O Prolog foi o suporte que escolhemos para o nível lógico da linguagem. Uma expressão, como $\log_e 4^2$, é representada a esse nível, por uma expressão lógica, do tipo $\log(e, \text{pwr}(2,4))$. O nível lógico é o nível computacional da linguagem. Os outros dois níveis (gráfico, e natural ou quasi-natural) são apenas níveis de apresentação dessa expressão. Por exemplo, a expressão "*logaritmo natural do quadrado de 4*", seria a forma natural[6] da expressão gráfica $\log_e 4^2$ ser lida ou interpretada, em linguagem corrente, por um ser humano. Essa é precisamente a forma que traduz, de forma inequívoca, o significado operacional dessa expressão.

### *2.2.1 A Expressão Lógica*

Ao nível lógico, todas as expressões matemáticas tratadas nesta tese são representadas por estruturas arbóreas binárias. As árvores binárias são uma forma bastante eficaz de se representar estruturas de expressões [VAL00]. Como veremos, também, as árvores binárias são também uma forma bastante eficiente de se lidar, computacionalmente, com expressões matemáticas. De facto, com base nesse tipo de estrutura, foi possível desenvolver, em Prolog, métodos bastante eficazes para lidar computacionalmente com a resolução simbólica de expressões matemáticas. A esse tipo de representação demos o nome de *expressão lógica*. Veremos, na Secção 2.4, que essas expressões exibem um tipo de estrutura que designamos por triangular.

**Definição 2.2-1** ***Expressão Lógica***

> *Uma expressão lógica é um termo, $s \in \mathcal{T}$, formado a partir de*

[6] A forma quasi-natural é para todos os efeitos uma forma abreviada da linguagem corrente. Tanto uma como a outra são formas inequívocas de se ler uma expressão matemática.

> *termos atómicos* $\mathcal{T}_{\mathcal{A}}$ *e de termos funcionais* $\mathcal{T}_{\mathcal{F}}$, *e onde* $\mathcal{T} = \mathcal{T}_{\mathcal{A}} \cup \mathcal{T}_{\mathcal{F}}$.

Termos como $4$, $x$, $\log(4,x)$ e $\log(e,\text{pwr}(2,4))$ são expressões lógicas. Nessas expressões, os termos $4$ e $x$, são termos atómicos e os termos $\log(4,x)$, $\log(e,\text{pwr}(2,4))$ e $\text{pwr}(2,4)$, termos funcionais.

**Definição 2.2-2** ***Termo Atómico***

> *Um termo atómico é um termo,* $s \in \mathcal{T}_{\mathcal{A}}$, *formado a partir dos números naturais* $\mathbb{N}$, *de constantes especiais,* $\mathcal{A} = \{x, -1, e\}$, *e onde* $\mathcal{T}_{\mathcal{A}} = \mathbb{N} \cup \mathcal{A}$. *As constantes* $\mathcal{A}$ *representam*
>
> *i)* *O número de Néper,* $e$,
>
> *ii)* *A variável matemática* $x$, *e*
>
> *iii)* *O sinal negativo,* $-1$.

Termos como $x$, $e$ ou $4$ são atómicos, mas termo $-4$, não. Nesta tese, foram apenas consideradas expressões matemáticas envolvendo uma única variável matemática $x$. A letra $x$ foi escolhida para representar esse termo. Em Prolog, um termo atómico é um termo simples.

**Definição 2.2-3** ***Termo Funcional***

> *Um termo funcional é um termo,* $f(s,t) \in \mathcal{T}_{\mathcal{F}}$, *binário, formado a partir de tipos funcionais* $\mathcal{O}p$, *de termos atómicos* $\mathcal{T}_{\mathcal{A}}$ *e de termos funcionais* $\mathcal{T}_{\mathcal{F}}$, *onde* $\mathcal{T}_{\mathcal{F}} = \mathcal{K} \cup \mathcal{F}$ *e*
>
> *i)* $\mathcal{K} = \{ f(s,t) \mid f \in \mathcal{O}p \wedge s,t \in \mathcal{T} \setminus \{x\} \}$,
>
> *ii)* $\mathcal{F} = \{ f(s,t) \mid f \in \mathcal{O}p \wedge s \in \{x\} \wedge t \in \mathcal{T} \vee s \in \mathcal{T} \wedge t \in \{x\} \}$

Termos como $\sqrt{x}$, $\log_e 4$ ou $\log_e \sqrt{x}$ são termos funcionais. Em Prolog, um termo funcional é um termo composto.

Nesta tese foram apenas incluídos os seguintes tipos funcionais:

| *Tipo Funcional* | *Operador* | *Tipo* |
|---|---|---|
| *sum* | Soma ou adição | $\mathcal{O}p^{sum} = \{sum, diff\}$ |
| *diff* | Diferença ou subtracção | |
| *prod* | Produto ou multiplicação | $\mathcal{O}p^{prod} = \{prod, div\}$ |
| *div* | Quociente ou divisão | |
| *pwr* | Potência ou potenciação | $\mathcal{O}p^{pwr} = \{pwr, root\}$ |
| *root* | Raiz ou radiciação | |
| *log* | Logaritmo | $\mathcal{O}p^{log} = \{log, exp\}$ |
| *exp* | Exponencial ou exponenciação | |
| *der* | Derivada ou derivação | $\mathcal{O}p^{der} = \{der\}$ |

onde

$$\mathcal{O}p = \mathcal{O}p^{sum} \cup \mathcal{O}p^{prod} \cup \mathcal{O}p^{pwr} \cup \mathcal{O}p^{log} \cup \mathcal{O}p^{der} =$$

$$= \{sum, diff, prod, div, pwr, root, log, exp, der\}$$

Como usualmente, numa expressão lógica, o operador principal é o operador mais à esquerda (*outermost*). O operador principal é, portanto, o operador com o âmbito mais alargado na expressão. Assim, numa expressão lógica, como $\log(e, \text{pwr}(2,4))$, o operador *log* é o operador principal e o operador *pwr*, um operador que designamos, apenas, por operador argumento. Um operador

argumento possui um âmbito mais restrito.

Numa expressão lógica, a componente principal só pode ser um termo atómico ou um termo funcional que define o seu operador principal. Por exemplo, numa expressão lógica, como $\log(e, \text{pwr}(2,4))$, a componente principal é o termo funcional $\log(e, \text{pwr}(2,4))$. Mas, numa expressão, como $\log(e, \text{pwr}(2, \text{root}(3,4)))$, a componente principal é também uma expressão com uma estrutura semelhante. O termo $\text{root}(3,4)$ é apenas um operador argumento da componente cujo operador principal é *pwr*. Isto leva à noção de tipo que é definido mais à frente.

Como veremos, na Secção 2.4, uma estrutura do tipo $\log(\square, \text{pwr})$ é uma estrutura que designamos por estrutura triangular básica do tipo *log_pwr*. A expressão $\log(e, \text{pwr}(2, \text{root}(3,4)))$ possui uma estrutura desse tipo. As expressões matemáticas, incluídas nesta tese, possuem estruturas triangulares.

### *2.2.2 A Constante Simbólica*

Numa expressão lógica, as constantes, variáveis e funções estão representadas por termos atómicos e funcionais. Apenas os números naturais, $\mathbb{N}$ e as constantes especiais $\mathcal{A}$ são representadas por termos atómicos. Todas as outras constantes e expressões são representadas por termos funcionais.

Nesta tese, apenas os números naturais, $\mathbb{N}$, são designados por constantes numéricas. Por exemplo, termos como $3$, $5$ ou $16$, representam constantes numéricas.

Os números inteiros negativos, $\mathbb{Z}^-$, estão representados por constantes simbólicas, $s \in \mathcal{I}^-$, onde

$$\mathcal{I}^- = \left\{ f(s,t) \in \mathcal{T} \mid f \in \{prod\} \wedge s \in \mathcal{S}^- \wedge t \in \mathbb{N} \right\}$$

e $\mathcal{S}^-$ representa o sinal negativo. Termos, como $\text{prod}(-1,3)$ ou $\text{prod}(-1,15)$ representam números inteiros negativos.

Os números inteiros, $\mathcal{I} = \mathbb{N} \bigcup \mathcal{I}^-$, são os únicas constantes, nesta tese, que podem intervir em cálculos aritméticos. Expressões, como $3(2)$ , $-3(5)$ ou $3+5$, podem ser transformadas em constantes numéricas ou simbólicas. O resultado dessas transformações são números inteiros. Mas, nem todas as constantes simbólicas, envolvendo números, são redutíveis por cálculo numérico.

**Exemplo 2.2-1** ***Calcular*** $3\left(\frac{4}{3}\right)$ ***por via axiomática***

*Expressões como* $3\left(\frac{4}{3}\right)$ *são apenas transformáveis por via axiomática, ou seja,*

$$3\left(\frac{4}{3}\right) \rightarrow \frac{12}{3} \rightarrow 4$$

---

Transformações desse tipo (Secção 2.5) envolvem a aplicação sistemática de um certo número de axiomas e/ou teoremas matemáticos elementares. De novo, nesta tese apenas focamos os axiomas que são aplicados no ensino secundário. Os axiomas e teoremas formam aquilo que designamos por base da reescrita axiomática, ou simplesmente de base axiomática do sistema de rescrita (Secção 4.3.1). A base axiomática, utilizada nesta tese, inclui apenas as propriedades matemáticas listadas no Apêndice B.

Como veremos, na Secção 4.3.1, o próprio cálculo numérico é efectuado, de uma forma que poderíamos classificar, também, de simbólica. Todos os resultados de uma resolução simbólica, sejam eles numéricos ou não, são obtidos por reescrita axiomática. Mas, nem todas as constantes simbólicas são transformáveis. Por

exemplo, expressões como $\log_e 2$ ou $\sqrt{2}$, são constantes simbólicas desse tipo. Essas expressões representam aquilo que, normalmente, é designado na literatura sobre reescrita [BAA98, DER01] por formas normais, ou irredutíveis. Mas como veremos na Secção 2.5, essas formas podem assumir qualquer tipo de padrão simbólico (termo atómico ou funcional).

**Definição 2.2-4** ***Constante Simbólica***

> *Uma constante simbólica é um termo funcional,* $f(s,t) \in \mathcal{K}$.

Os seguintes elementos estão representados por constantes simbólicas:

- O sinal negativo de uma expressão, ou seja, um elemento do conjunto

$$\mathcal{S}^- = \left\{ f(s,t) \in \mathcal{T} \mid f \in \{prod\} \wedge s \in \mathcal{S}^- \wedge t \in \{1\} \right\}$$

- Números inteiros negativos, ou seja, um elemento do conjunto

$$\mathcal{I}^- = \left\{ f(s,t) \in \mathcal{T} \mid f \in \{prod\} \wedge s \in \mathcal{S}^- \wedge t \in \mathbb{N} \setminus \{0\} \right\}$$

- Números racionais, $p/q$, ou seja, um elemento do conjunto

$$\mathcal{Q} = \left\{ f(s,t) \in \mathcal{T} \mid f \in \{div\} \wedge s \in \mathbb{N} \bigcup \mathcal{I}^- \wedge t \in \mathbb{N} \bigcup \mathcal{I}^- \setminus \{0\} \right\}$$

- A raiz de ordem par $n$ de um número, $\sqrt[n]{p}$, ou seja, um elemento do conjunto

$$\bar{\mathcal{Q}}_1^{root} = \left\{ f(s,t) \in \mathcal{T} \mid f \in \{root\} \wedge s = 2k \wedge k \in \mathbb{N} \setminus \{0\} \wedge t \in \mathbb{N} \right\}$$

- A raiz de ordem ímpar $n$ de um número, $\sqrt[n]{p}$, ou seja, um elemento do conjunto

$$\bar{\mathcal{Q}}_2^{root} = \{ f(s,t) \in \mathcal{T} \mid f \in \{root\} \wedge s = 2k+1 \wedge k \in \mathbb{N} \setminus \{0\} \wedge t \in \mathbb{N} \cup \mathcal{I}^- \}$$

- O logaritmo de base $a$ de um número, $\log_a p$, ou seja, um elemento do conjunto

$$\bar{\mathcal{Q}}^{log} = \{ f(s,t) \in \mathcal{T} \mid f \in \{log\} \wedge s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{0\} \}$$

- A exponencial de base $a$ de um número, $a^p$, ou seja, um elemento do conjunto

$$\bar{\mathcal{Q}}^{exp} = \{ f(s,t) \in \mathcal{T} \mid f \in \{exp\} \wedge s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \}$$

Como veremos na Secção 4.2.1, o termo, $-1$, possui um significado especial. O seu uso só é permitido em determinados contextos.

**Exemplo 2.2-2** ***Representação lógica de*** $-6$

*A constante* $-6$*, lida como "menos 6" ou "-6", é representada pelo termo* $\text{prod}(-1,6)$*. A expressão* $-(-6)$*, lida como "menos menos 6", é representada por* $\text{prod}(-1,\text{prod}(-1,6))$*. Porém, essa expressão, lida como "produto de -1 por -6", seria representada por* $\text{prod}(\text{prod}(-1,1),\text{prod}(-1,6))$.

---

Uma expressão do tipo $\text{prod}(-1,\_)$, é designada por simétrica de $\_$. Por simetria, o inverso é também verdadeiro. Neste tipo de expressões, o sinal está representado de forma explícita. Os números inteiros negativos estão representados desta forma.

### *2.2.3 Os Vários Tipos de Expressões*

As expressões matemáticas, apresentadas nesta tese, são representadas por expressões lógicas. Como sabemos, numa expressão lógica, podem estar presentes, zero ou mais operadores matemáticos. Expressões, como $\log_e 4^2$ e $(\log_e 4)^2$, são representadas por expressões lógicas como $\log(e, \text{pwr}(2,4))$ e $\text{pwr}(2, \log(e,4))$, respectivamente. Como se pode observar, nessas expressões estão presentes os mesmos dois operadores matemáticos, mas numa ordem diferente. Uma leitura simbólica dessas representações, ou seja, as expressões "*log e pwr 2 4*" e "*pwr 2 log e 4*", mostra, de facto, que essas duas expressões diferem, apenas, na ordem em que os respectivos operadores estão dispostos. Esse tipo de leitura confere à expressão aquilo que designamos por significado operacional dessa expressão. Como veremos na Secção 2.3.1, as expressões matemáticas podem ser caracterizadas com base nesse tipo de leitura.

Mas o que se poderia dizer do significado operacional de expressões como $\log_e 4^2$ e $\log_e 2^4$, onde os operadores estão dispostos na mesma ordem? Uma leitura simbólica das respectivas representações lógicas, ou seja, as expressões "*log e pwr 2 4*", e "*log e pwr 4 2*", mostraria que essas duas expressões possuem, de facto, o mesmo significado operacional. Tanto uma com a outra, possuem estruturas do tipo $\log(\square, \text{pwr})$. Poderíamos dizer que uma estrutura desse tipo define, de facto, um certo tipo de expressão. E, como é óbvio, expressões do mesmo tipo devem possuir características operacionais semelhantes. Relativamente aos axiomas utilizados nesta tese, ambas são tratadas, simbolicamente, de forma semelhante, ou seja, a sua base axiomática (axiomas e teoremas) é a mesma.

Como veremos, na Secção 2.4.1, esse tipo de estrutura possui uma assinatura bem definida, através da qual se podem caracterizar as expressões matemáticas. Porém, expressões, como 2 ou $x$, são de um tipo que designamos por atómico e que, como veremos mais adiante, não possui assinatura.

**Definição 2.2-5** ***Tipo Atómico***

> *Um tipo atómico é uma entidade sintáctica, denotada por* $none$ *, que identifica um termo atómico. O tipo* $none$ *é representado, abstractamente, pelo símbolo* $\square$ *.*

De acordo com essa definição, todos os termos atómicos são do mesmo tipo (átomos). Uma expressão, como $\log_e 4^2$, possui, portanto, três termos atómicos: "$e$", 4 e 2, todos do tipo $none$. Mas essa expressão possui uma estrutura de um tipo que designamos por $\log(\square, \text{pwr})$. Os símbolos $log$ e $pwr$, são tipos que designamos por funcionais.

**Definição 2.2-6** ***Tipo Funcional***

> *Um tipo funcional é um símbolo de função,* $f \in \mathcal{O}p$ *, que define o tipo de operador matemático. Um tipo funcional é representado, abstractamente, por* $op(\_,\_)$ *, onde o símbolo* $\_$ *pode ser um tipo atómico ou funcional.*

De acordo com esta definição, expressões como $\log_e 4^2$ possuem também dois termos funcionais do tipo $log$ e $pwr$. Como sabemos, essa expressão é representada por uma expressão lógica do tipo $\log(e, \text{pwr}(2,4))$, com duas componentes: uma, como uma estrutura lógica do tipo $\log(\square, \text{pwr})$, e uma outra, com uma estrutura lógica do tipo $\text{pwr}(\square,\square)$. Como veremos na Secção 2.4.1, essa expressão é caracterizada como sendo uma expressão lógica com uma estrutura do tipo $\log(\square, \text{pwr})$ e uma assinatura $log\_pwr$.

Mas, expressões, como $-x$ ou $\log_e 2$, possuem apenas um tipo funcional. A expressão $-x$, por exemplo, possui uma estrutura lógica do tipo $\text{prod}(\square,\square)$, e a expressão $\log_e 2$, uma estrutura lógica do tipo $\log(\square,\square)$. Como veremos na Secção

2.4.1, tanto uma como a outra são expressões que caracterizamos como possuindo uma assinatura do tipo *op* .

O facto de todos os termos atómicos serem do mesmo tipo, permite tratar expressões, como $\log_e 4^2$, $\log_e 2^4$ ou $\log_{10} x^2$, como sendo do mesmo tipo, ou seja, expressões com uma estrutura lógica do tipo $\log(\square, \text{pwr}(\square,\square))$.

Tanto os tipos atómicos como os funcionais são tipos que consideramos básicos. Como vimos atrás, uma expressão lógica pode conter um ou mais destes tipos. Como poderíamos então definir o tipo de uma expressão?

**Definição 2.2-7** ***Tipo de Expressão***

> *O tipo de expressão é uma entidade sintáctica formada pela concatenação de todos os seus tipos atómicos e funcionais, separados por* _ *que identifica a estrutura da expressão.*

De acordo com esta definição, uma expressão, como $\log_e \sqrt{4^3}$, seria do tipo "*log_none_root_none_pwr_none_none*". Como veremos na Secção 2.4, uma expressão desse tipo possui uma assinatura *log_root* . Como se pode observar, a assinatura de uma expressão inclui apenas os tipos funcionais da sua componente principal. Uma expressão, como $\log_e \sqrt[3]{4^2}$, seria também do mesmo tipo.

## 2.3 A Leitura Operacional de Expressões

O que acontece, então, quando tentamos ler uma expressão matemática? Consideremos, por exemplo, a expressão $x+3$. Que significado lhe poderíamos atribuir? Para já, não se trata de nenhuma palavra, ou frase, escrita em Português. Não poderíamos caracterizá-la como tal. Quanto muito poderíamos dizer que a

expressão, em questão, é formada de elementos mais simples, como por exemplo, os símbolos "$x$", "$+$" e "3". Em Português, essa expressão não teria qualquer significado. Em Matemática, porém, o seu significado seria bastante preciso. Operacionalmente, ela significaria "*a soma de* $x$ *com* 3". A sua leitura, em Português, só poderia ser feita, símbolo a símbolo, ou seja, de uma forma que consideraríamos como, meramente, simbólica.

**Definição 2.3-1** ***Leitura Simbólica***

> *Uma leitura simbólica é uma forma de leitura que se faz, símbolo a símbolo, da esquerda para a direita, excluindo quaisquer símbolos estruturantes, como parênteses ou vírgulas*[7].

Para se *ler* uma expressão, como $x+3$, em Português, teríamos de o fazer, símbolo a símbolo, ou seja, interpretá-la como se, de facto, significasse o mesmo que "*x mais 3*". Mas essa não seria, certamente, a forma mais correcta de o fazer. Qualquer um, com algum conhecimento de Matemática, facilmente reconheceria que essa expressão tem um significado Matemático e que esse significado é, de facto, "*soma de x com 3*". Escrita dessa forma, não existiriam quaisquer dúvidas de que a expressão $x+3$ representa, de facto, uma frase, em Português, sintáctica e semanticamente, correcta.

Mas uma leitura simbólica nem sempre traduz o verdadeiro significado operacional da expressão. Leituras desse tipo podem até levantar algumas questões de coerência ou, mesmo, de interpretação, como veremos a seguir.

- De coerência porque muitas dessas expressões matemáticas não podem ser lidas dessa forma. Por exemplo, uma expressão como $4^2$ não pode ser lida de uma forma simbólica. Qual seria a leitura do símbolo $2$,

[7] Os parênteses e as vírgulas são apenas utilizados para evidenciar uma determinada estrutura, ou desambiguá-la.

nessa expressão?

- De interpretação porque essa forma de leitura pode conduzir, em alguns casos, a interpretações ambíguas. Por exemplo, uma expressão, como $\log_e(x+1)$, teria uma leitura simbólica, no mínimo, ambígua.

**Exemplo 2.3-1** ***Leitura simbólica de*** $2(2+3)$ ***e*** $2(2)+3$

*A expressão* $2(2+3)$*, lida, de uma forma simbólica, como "2 vezes 2 mais 3", seria ambígua, pois levaria à mesma interpretação, duas expressões que sabemos ser, matematicamente, diferentes, ou seja, as expressões* $2(2+3)$ *e* $2(2)+3$.

---

Para as desambiguar, teríamos de incluir, nessa leitura, os próprios parênteses, ou seja, lê-la como "2 *vezes abrir parênteses 2 mais 3 fechar parênteses*"? Essa frase não faria qualquer sentido, tanto em Português como em Matemática. Seria, quanto muito, uma mera soletração dos seus símbolos!

### *2.3.1 O Significado Operacional*

Uma leitura simbólica confere apenas um significado simbólico à expressão. Esse significado, por vezes, pode ser ambíguo. Para que uma expressão possa ser lida de uma forma inequívoca, é necessário que a expressão esteja representada também de uma forma inequívoca. A representação lógica dá-nos essa pista.

**Definição 2.3-2** ***Leitura Operacional***

> *A leitura operacional de uma expressão é simplesmente uma leitura simbólica da sua representação lógica.*

Uma leitura operacional confere à expressão um significado, em Português,

correcto e, totalmente, inequívoco da expressão. Poderíamos mesmo dizer que uma leitura operacional confere um *significado operacional*. A forma correcta de se ler, ou interpretar, uma expressão, como $2(2+3)$ (Exemplo 2.3-1), seria *lê-la*, como "*o produto de 2 pela soma de 2 com 3*". Essa leitura não seria mais do que uma leitura simbólica da sua expressão lógica, $\text{prod}(2,\text{sum}(2,3))$. Lida dessa forma, a expressão seria, totalmente, inequívoca e representaria, única e exclusivamente, a expressão $2(2+3)$. O seu significado, em Português, seria também, totalmente, inequívoco. A expressão $2(2)+3$ seria lida também, de uma forma inequívoca, como "*a soma do produto de 2 por 2 com 3*".

Leituras operacionais são totalmente coerentes com a forma como as expressões são lidas, ou escritas, em linguagem corrente.

**Exemplo 2.3-2** ***Leitura operacional de*** $4^2$

> *Uma expressão, como* $4^2$*, lida operacionalmente como "o quadrado de 4", "a potência de grau 2 de 4" ou simplesmente "pwr 2 4", é totalmente inequívoca.*

---

Como se pode observar, numa leitura operacional são apenas utilizadas formas prefixas dos operadores matemáticos. Símbolos, como "*mais*" ou "*menos*", representam apenas uma leitura, meramente, simbólica dos operadores infixos "$+$" ou "$-$", respectivamente.

A leitura operacional fornece também uma forma de interpretação que poderíamos classificar de operacional ou computacional. Por exemplo, uma expressão lógica, como $\log(e,\text{pwr}(2,4))$, seria interpretada, computacionalmente, como sendo a expressão matemática $\log_e 4^2$. Escrita como o "*logaritmo natural do*

*quadrado de 4*" ou "*log e pwr 2 4*", seria lida, operacionalmente, como $\log(e, \text{pwr}(2,4))$.

Numa leitura operacional, os operadores matemáticos são lidos em profundidade (*depth-first*) e da esquerda para a direita (*left-right*).

### *2.3.2 Os Operadores Matemáticos*

As expressões matemáticas possuem elementos que designamos, normalmente, por operadores. Os operadores podem ser representados de uma forma implícita, ou explícita. Na sua forma explícita, os operadores podem ainda ser representados de uma forma infixa ou prefixa.

- **Infixa** – O operador é escrito entre os seus argumentos. Por exemplo, numa expressão, como $2+3$, o operador soma ("$+$") está representado na sua forma infixa;
- **Prefixa** – O operador é escrito antes dos seus argumentos. Por exemplo, numa expressão, como $\log_{10} 3$, o operador logaritmo ("$\log$") está representado na sua forma prefixa.

A leitura de uma expressão não depende apenas da forma como ela está escrita. Depende também da forma como a expressão é interpretada. A utilização de formas implícitas ou infixas, para representar os operadores, conduzem, normalmente, a leituras simbólicas, ou interpretações, ambíguas. A leitura simbólica de uma expressão está, normalmente, associada à forma como a expressão está escrita. É um tipo de leitura que poderíamos designar como uma mera "*tradução*", daquilo que está escrito. A leitura operacional, por outro lado, está normalmente associada à forma como a expressão deve se interpretada. É um tipo de leitura que poderíamos designar como uma "*interpretação*" daquilo que se está a ler.

Qualquer expressão, onde esses dois tipos de leitura (simbólica e operacional) sejam idênticos, não suscitaria qualquer tipo de dúvida, quanto à sua interpretação. Por exemplo, uma expressão, escrita como "*a soma de x com 3*", seria correctamente interpretada como a soma dos elementos "$x$" e "3". Diríamos, nesse caso, que a expressão "*se lê tal como se escreve*". Essa seria a forma correcta de transmitir o significado operacional da expressão.

Podemos dizer que a leitura simbólica de expressões matemáticas, onde estejam presentes, um ou mais, operadores infixos, é, em geral, ambígua.

**Exemplo 2.3-3** ***Leitura simbólica de*** $(1-2)+3$ ***e*** $1-(2+3)$

*Expressões, como* $(1-2)+3$ *e* $1-(2+3)$*, seriam lidas, simbolicamente, como "1 menos 2 mais 3". Essa não seria, certamente, a forma correcta de as representar, pois a sua leitura seria ambígua.*

---

O mesmo sucederia com muitos outros operadores infixos, como por exemplo, o operador quociente, ou a forma infixa de operadores implícitos, como por exemplo o produto.

A leitura simbólica de expressões matemáticas, onde estejam presentes apenas operadores prefixos, binários (ou com aridade bem determinada), é sempre inequívoca. De facto, numa representação desse tipo, os operadores são lidos antes dos seus argumentos, que são em número bem determinado, sendo portanto possível saber-se qual é o operador que está associado a qualquer argumento. Por exemplo, representando as expressões mencionadas no Exemplo 2.3-3, como $\text{sum}(\text{diff}(1,2),3)$ e $\text{diff}(1,\text{sum}(2,3))$, tornaria possível a sua leitura simbólica de uma forma correcta. Representadas nessa forma, as expressões $(1-2)+3$ e $1-(2+3)$ seriam interpretadas operacionalmente de uma forma inequívoca.

Mas a forma prefixa que é escolhida para representar um operador, nem sempre é consensual. Por exemplo, há quem utilize formas prefixas, de diferente aridade[8], para representar expressões, como $\sqrt{2}$ ou $\sqrt[3]{2}$. A forma unária, $\text{sqrt}(2)$, é ainda utilizada como uma forma prefixa de representar a expressão $\sqrt{2}$. Nessa forma de representação, o grau da raiz está definido de uma forma implícita. Para se representar uma expressão como $\sqrt[3]{2}$, na sua forma prefixa, seriam necessários, pelo menos, dois argumentos. A forma binária seria a mais consensual pois satisfaria a ambas. Um dos argumentos seria utilizado precisamente para definir o grau da raiz. A expressão $\sqrt{2}$ seria representada, neste caso, por uma expressão do tipo $\text{root}(2,2)$, binária. O mesmo se poderia dizer do operador logaritmo.

Nesta tese, para que uma leitura seja, totalmente, inequívoca, é necessário que todos os operadores, presentes na expressão, estejam representados na sua forma prefixa e binária.

**Exemplo 2.3-4** ***Leitura simbólica de*** $\text{root}(2,2)+3$ ***e*** $\text{root}(2,2+3)$

> *Expressões como* $\text{root}(2,2)+3$ *e* $\text{root}(2,2+3)$*, com operadores infixos e prefixos, seriam lidas, simbolicamente, como a "raiz de grau 2 de 2 mais 3"*[9]*, ou seja, de uma forma ambígua.*

---

Optámos, nesta tese, por representar os operadores matemáticos numa forma prefixa e binária. Essa forma de representação levou-nos ao desenvolvimento de uma forma computacional de representar e lidar com expressões matemáticas. Uma forma de representação que designámos por estrutura triangular.

---

[8] A aridade de uma relação define o número de argumentos envolvidos nessa relação. Uma relação unária seria, portanto, uma relação de aridade 1.

[9] Numa representação lógica com operadores infixos, os operadores $+$ e $-$, seriam lidos como "*mais*" e "*menos*", respectivamente.

## 2.4 A Estrutura Triangular de Expressões

Como vimos atrás, uma leitura operacional é totalmente inequívoca. O facto de a estrutura lógica de uma expressão matemática ser arbórea e dos operadores matemáticos estarem representados de uma forma prefixa e binária, tornou possível uma forma de leitura operacional que designámos por triangular. Por exemplo, uma expressão como $\text{sum}(1,\text{sum}(2,3))$ pode ser lida, simbolicamente, como uma composição lógica de duas estruturas triangulares: $\text{sum}(\square,\text{sum})$ e $\text{sum}(\square,\square)$. Essa forma de leitura foi motivada por uma análise exaustiva e detalhada dos axiomas e teoremas que teríamos de utilizar na resolução simbólica de expressões. Todos os axiomas e teoremas estudados, nesta tese, possuem estruturas lógicas que designámos por estruturas triangulares básicas.

**Definição 2.4-1** ***Estrutura Triangular Básica***

> *Uma estrutura triangular básica é um tipo de estrutura arbórea binária, denotada por* $\Xi\text{-}\mathcal{Sig}$ *, cujo nível estrutural é* $\mathcal{N}(\Xi\text{-}\mathcal{Sig})$ *e cuja assinatura é* $\mathcal{Sig}$*.* $\Xi\text{-}\mathcal{Sig}$ *é o conjunto de todas as estruturas triangulares básicas com assinaturas* $\mathcal{Sig}$ *, ou seja, o conjunto*
>
> $$\Xi\text{-}\mathcal{Sig} = \{\Xi\text{-}a \mid a \in \mathcal{Sig}\}$$

Uma estrutura lógica do tipo $\text{div}(\square,\square)$ é um tipo de estrutura que designamos por estrutura triangular básica do tipo *div*, ou seja, uma estrutura do tipo $\Xi\text{-}div$. Expressões como $\frac{2}{3}$ possuem estruturas lógicas desse tipo. A estrutura $\Xi\text{-}div$ está ilustrada na Figura 2.4.1.

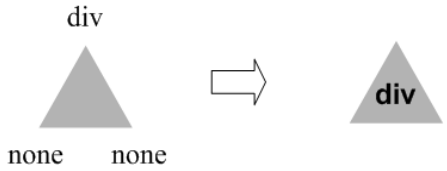



**Figura 2.4.1** *Estrutura triangular de nível 1*

**Definição 2.4-2** ***Estrutura Triangular*** $\Xi$-*op*

> *Uma estrutura, do tipo* $\Xi$-*op* , *é uma estrutura triangular básica com apenas um único termo funcional de tipo* $op \in \mathcal{O}p$ , *ou seja, uma estrutura do tipo* $op(\square,\square)$.

Todos os operadores matemáticos, apresentados nesta tese, possuem estruturas desse tipo. Uma estrutura do tipo $\Xi$-*op* é uma estrutura triangular básica de nível 1, ou seja, uma estrutura do tipo $(\mathrm{N}1)_{op}$. Uma estrutura de nível 1, denotada por $\mathrm{N}1$, possui apenas um tipo funcional *op* .

**Definição 2.4-3** ***Nível Estrutural***

> *O nível de uma estrutura triangular básica,* $\mathcal{N}(\Xi\text{-}\mathcal{S}ig)$, *é o número de termos funcionais presentes nessa estrutura, ou seja, o número dado por* $\mathcal{N} : \Xi\text{-}\mathcal{S}ig \to \mathbb{N}$. *O nível estrutural é denotado por* $\mathrm{N}i$, *onde* $i = 0,1,2,3$.

**Exemplo 2.4-1** ***Estrutura*** $\Xi$-*div*

*Expressões como* $\frac{2}{3}$ *são estruturas de nível 1.*

---

Uma estrutura do tipo $\log(\square,\mathrm{pwr})$ é uma estrutura que designamos por

estrutura triangular básica do tipo *log_pwr*, ou seja, uma estrutura do tipo Ξ-*log_pwr*. Expressões, como $\log_e 4^2$, possuem estruturas desse tipo. Esse tipo de estrutura está ilustrado na Figura 2.4.2.

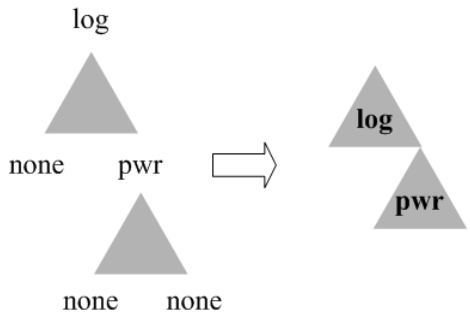



**Figura 2.4.2** *Estrutura triangular de nível 2*

**Definição 2.4-4** ***Estrutura Triangular*** Ξ-*op_op*

> *Uma estrutura, do tipo* Ξ-*op_op*, *é uma estrutura triangular básica com apenas dois termos funcionais de tipo* $op \in \mathcal{O}p$, *ou seja, uma estrutura do tipo* $op(op,\square)$ *ou* $op(\square,op)$.

Uma estrutura do tipo Ξ-*op_op* é uma estrutura triangular básica de nível 2, ou seja, uma estrutura do tipo $(\mathrm{N2D})_{op\_op}$ ou do tipo $(\mathrm{N2E})_{op\_op}$. Uma estrutura de nível 2 possui apenas dois tipos funcionais *op*. Uma estrutura do tipo $(\mathrm{N2D})_{op\_op}$ é uma estrutura triangular básica de nível 2 com ramificações à direita ( N2D ), ou seja, uma estrutura do tipo $op(\square,op)$. Uma estrutura de tipo $(\mathrm{N2E})_{op\_op}$, por outro lado, é uma estrutura triangular de nível 2 com ramificações à esquerda ( N2E ), ou seja, uma estrutura do tipo $op(op,\square)$.

Expressões como $\log_e 4^2$ possuem, portanto, estruturas compostas do tipo $(\mathrm{N2D})_{log\_pwr}(\mathrm{N1})_{pwr}$. A sua componente principal é uma expressão do tipo

$(\mathrm{N2D})_{log\_pwr}$, ou seja, uma expressão com uma estrutura lógica do tipo $\log(\square,\mathrm{pwr})$. A outra componente é uma expressão do tipo $(\mathrm{N1})_{pwr}$, localizada à sua direita e com uma estrutura do tipo $\mathrm{pwr}(\square,\square)$.

Uma estrutura lógica do tipo $\mathrm{sum}(\mathrm{sum},\mathrm{sum})$ é um tipo de estrutura que designamos por estrutura triangular básica de tipo *sum_sum_sum*, ou seja, uma estrutura do tipo Ξ-*sum_sum_sum*. Expressões como $(x+2)+(x+4)$ possuem estruturas desse tipo.

**Definição 2.4-5** ***Estrutura Triangular*** Ξ-*op_op_op*

> *Uma estrutura, do tipo* Ξ-*op_op_op*, *é uma estrutura triangular básica com três termos funcionais de tipo* $op \in \mathcal{O}p$, *ou seja, uma estrutura do tipo* $op(op,op)$.

Uma estrutura do tipo Ξ-*op_op_op* é uma estrutura triangular básica de nível 3, ou seja, uma estrutura do tipo $(\mathrm{N3})_{op\_op\_op}$. Uma estrutura de nível 3 possui três tipos funcionais *op*. Estruturas desse tipo são estruturas com ramificações, tanto à esquerda como à direita, ou seja, estruturas do tipo $op(op,op)$. Esse tipo de estrutura está ilustrado na Figura 2.4.2.

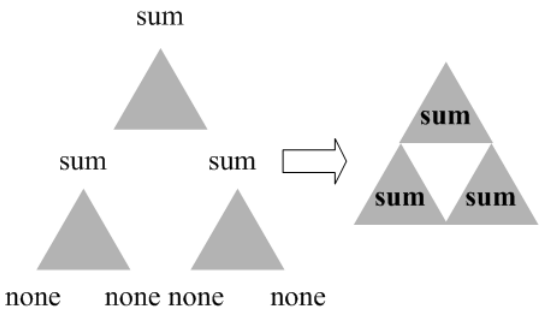



**Figura 2.4.3** ***Estrutura triangular de nível 3***

Expressões como $(x+2)+(x+4)$ possuem estruturas compostas do tipo $(\mathrm{N3})_{sum\_sum\_sum}(\mathrm{N1})_{sum}(\mathrm{N1})_{sum}$. A sua componente principal é uma expressão do tipo $(\mathrm{N3})_{sum\_sum\_sum}$, ou seja, uma expressão com uma estrutura lógica do tipo $\mathrm{sum}(\mathrm{sum},\mathrm{sum})$. As duas outras componentes são expressões do tipo $(\mathrm{N1})_{sum}$ localizadas à sua esquerda e à sua direita e com uma estrutura do tipo $\mathrm{sum}(\square,\square)$.

Uma estrutura do tipo $\square$ é um tipo de estrutura que designamos por estrutura triangular básica do tipo *none*, ou seja, uma estrutura do tipo Ξ-*none*. Expressões, como 2, 4 ou $x$, possuem estruturas desse tipo.

**Definição 2.4-6** ***Estrutura Triangular*** Ξ-*none*

> *Uma estrutura, de tipo* Ξ-*none é uma estrutura triangular básica sem nenhum termo funcional, ou seja, uma estrutura do tipo* $\square$.

Uma estrutura de tipo Ξ-*none* é uma estrutura triangular básica de nível 0, ou seja, uma estrutura do tipo $(\mathrm{N0})_{none}$. Uma estrutura triangular de nível 0 não possui nenhum tipo funcional. Estruturas deste tipo possuem apenas um termo atómico. Esse tipo de estrutura está ilustrado na Figura 2.4.4.

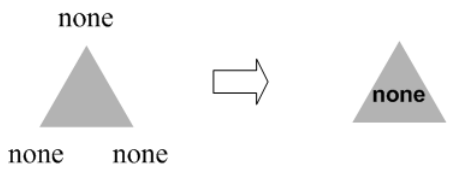



**Figura 2.4.4** ***Estrutura triangular de nível 0***

Existem apenas quatro (4) níveis estruturais possíveis, denotados por $\mathrm{N}i$, $i=0,1,2,3$. Esses níveis estão representados na Figura 2.4.5. Com excepção de $\mathrm{N2}$, todos eles correspondem a uma única estrutura triangular básica. No caso do nível $\mathrm{N2}$, existem duas estruturas básicas: uma à esquerda, $\mathrm{N2E}$ e outra à direita, $\mathrm{N2D}$.

Existem, portanto, quatro (4) níveis estruturais: N0, N1, N2 e N3, e cinco (5) estruturas triangulares básicas: $(N0)_{none}$, $(N1)_{op}$, $(N2E)_{op\_op}$, $(N2D)_{op\_op}$ e $(N3)_{op\_op\_op}$. Esse número de estruturas corresponde, precisamente, ao número de combinações possíveis que se podem formar com os quatro (4) níveis estruturais, tendo em conta que em cada uma dessas estruturas só podem estar presentes termos atómicos □, ou termos binários do tipo $op(\_,\_)$, e onde _ denota qualquer um desses termos. As cinco (5) estruturas básicas estão ilustradas, também, na Figura 2.4.5.

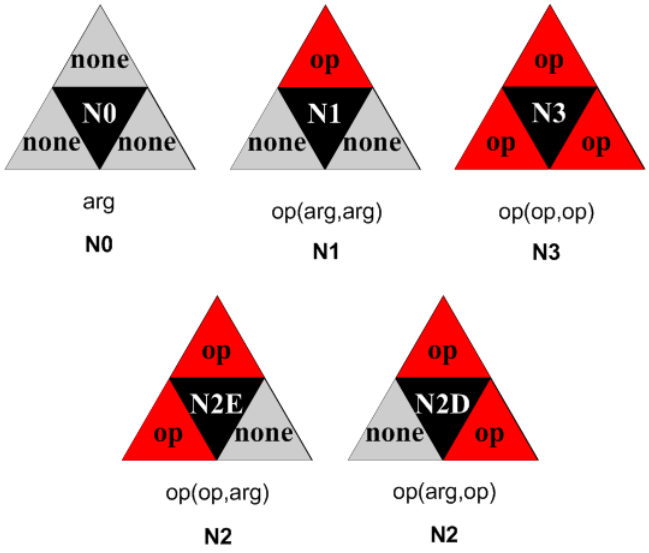



**Figura 2.4.5** *Estruturas triangulares básicas*

Uma estrutura do tipo $op(op(op(\_,\_),\_),\_)$ não é uma estrutura triangular básica. O terceiro termo funcional, $op(\_,\_)$, faz parte da estrutura triangular de um dos seus argumentos, $op(op(\_,\_),\_)$. Trata-se, efectivamente, de uma estrutura triangular do tipo $(N2E)_{op\_op}$. Como veremos, mais adiante, todas essas estruturas se podem formar a partir das cinco (5) estruturas básicas.

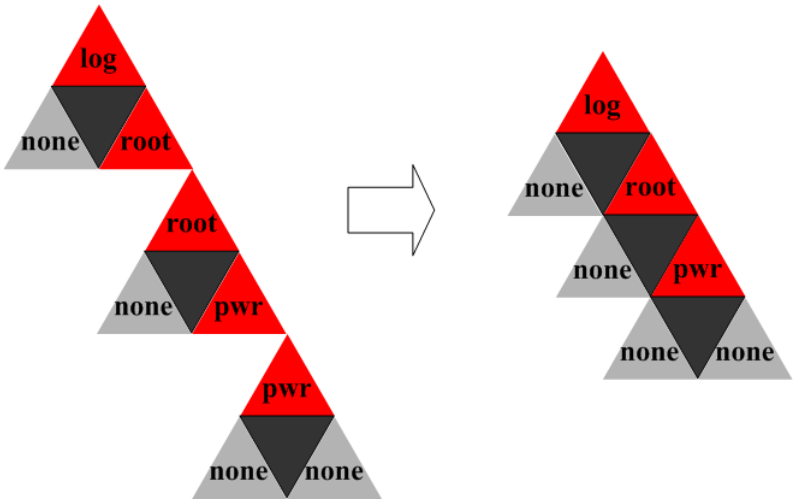
Todas as expressões matemáticas, apresentadas nesta tese, possuem estruturas que são uma composição lógica das cinco (5) estruturas triangulares

básicas. Especulamos que a maioria das expressões matemáticas, usuais, sejam deste tipo. No entanto, é de se esperar, que domínios mais específicos como, por exemplo, o dos polinómios, sejam necessários contextos mais complicados.

**Exemplo 2.4-2** ***Estrutura triangular de*** $\log_e \sqrt{x^3}$

*Uma expressão, como* $\log_e \sqrt{x^3}$, *possui uma estrutura do tipo* $(\mathrm{N2D})_{log\_root}(\mathrm{N2D})_{root\_pwr}(\mathrm{N1})_{pwr}$, *ou seja, uma estrutura lógica do tipo* $\log(\square,\mathrm{root}(\square,\mathrm{pwr}(\square,\square)))$, *que é uma composição lógica de três (3) estruturas triangulares básicas:* $\log(\square,\mathrm{root})$, $\mathrm{root}(\square,\mathrm{pwr})$ *e* $\mathrm{pwr}(\square,\square)$. *A sua componente principal é, neste caso, a componente com a estrutura* $\log(\square,\mathrm{root})$, *ou seja a componente com a assinatura* log_root .

Essa composição está ilustrada na Figura 2.4.6.




$$\underbrace{\underbrace{log\_none\_root}_{(\mathrm{N2D})_{log\_root}}\_\underbrace{root\_none\_pwr}_{(\mathrm{N2D})_{root\_pwr}}\_\underbrace{pwr\_none\_none}_{(\mathrm{N1})_{pwr}}}_{\log(\square,\mathrm{root}(\square,\mathrm{pwr}(\square,\square)))}$$

$\log(\square,\mathrm{root})$ $\quad$ $\mathrm{root}(\square,\mathrm{pwr})$ $\quad$ $\mathrm{pwr}(\square,\square)$

**Figura 2.4.6** *Composição lógica de estruturas*

Como veremos, as expressões matemáticas, apresentadas nesta tese, ficam completamente caracterizadas pela estrutura triangular da sua componente principal. Por exemplo, uma expressão como $\log_e \sqrt{x^3}$ (Exemplo 2.4-2) é caracterizada como uma expressão do tipo *log_root* , ou seja, uma expressão com uma estrutura lógica do tipo $\log(\square, \text{root})$ . A componente principal está ilustrada na Figura 2.4.7.

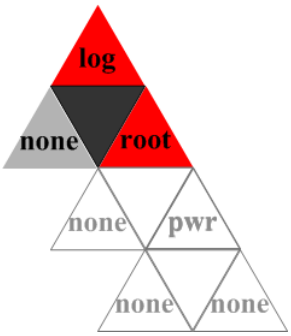



**Figura 2.4.7** *Componente principal*

As expressões matemáticas, apresentadas nesta tese, possuem um número finito de componentes, isto é, a sua estrutura lógica é finita. Por exemplo, uma expressão como $\log_e \sqrt{4^3}$ (Exemplo 2.4-2), possui três componentes. A sua componente principal é uma expressão do tipo $(\text{N2D})_{log\_root}$ , ou seja, uma expressão com uma estrutura lógica do tipo $\log(\square, \text{root})$ . A expressão $\log_e \sqrt{4^3}$ é, portanto, caracterizada como uma expressão do tipo *log_root* .

Como veremos na Secção 4.4.1, a resolução simbólica é efectuada ao nível das componentes e, portanto, a aplicação de estratégias depende do conhecimento que se tem sobre as estruturas triangulares dessas componentes. Como é óbvio, expressões com leituras operacionais diferentes possuem estruturas triangulares

diferentes. Por exemplo, uma expressão, como $x+2+x$, pode ser lida, operacionalmente, de duas formas diferentes e representada, portanto, como uma expressão lógica $\text{sum}(x,\text{sum}(2,x))$ ou $\text{sum}(\text{sum}(x,2),x)$. Mas mesmo aquelas com uma única forma de leitura podem ser resolvidas, simbolicamente, de formas diferentes. As várias formas de resolução dependem apenas da ordem em que as componentes são tratadas, ou seja, do tipo de estratégia aplicada. Por exemplo, expressões, como $\log_e \sqrt{4^3}$, podem ser resolvidas, simbolicamente, como logaritmos ou logaritmos de raízes. Como veremos na Secção 4.4.1, a aplicação de uma estratégia de conversão levaria à resolução da sua componente principal *log_root*, enquanto que a aplicação de uma estratégia de simplificação levaria à resolução da sua componente *root_pwr*.

### *2.4.1 O Conceito de Assinatura*

Nesta tese, a assinatura de uma expressão é determinada, única e exclusivamente, pela estrutura triangular da sua componente principal. A sua estrutura é determinada pela forma como é lida.

**Exemplo 2.4-3** ***Assinatura de*** $\log_e 4^2$**,** $\log_e (x+4)^2$ ***e*** $\log_e \left(\sqrt{x+4}\right)^2$

*Uma expressão, como* $\log_e 4^2$*, lida como o "logaritmo de uma potência", seria representada por uma expressão lógica do tipo* $\log(\square,\text{pwr})$*. A sua assinatura seria, portanto,* log_pwr*. Expressões, como* $\log_e (x+4)^2$ *ou* $\log_e \left(\sqrt{x+4}\right)^2$*, teriam todas a mesma assinatura. Todas elas possuem componentes principiais com a mesma estrutura.*

---

O mesmo se poderia dizer de expressões, como $\text{sum}(x,\text{sum}(2,x))$ ou

$\text{sum}(\text{sum}(x,2),x)$. Só que, neste caso, as componentes principais possuem estruturas triangulares do mesmo nível mas com orientações diferentes, ou seja, estruturas $(\text{N2E})_{sum\_sum}$ e $(\text{N2D})_{sum\_sum}$ com a mesma assinatura *sum_sum*.

**Definição 2.4-7** ***Assinatura***

> *A assinatura de uma expressão, $s \in \mathcal{S}ig$, é simplesmente a leitura simbólica da estrutura triangular da sua componente principal. $\mathcal{S}ig$ é o conjunto de todas as assinaturas, ou seja, o conjunto*
>
> $$\mathcal{S}ig = \{ f, f\_g, f\_g\_h \mid f, g, h \in \mathcal{O}p \}$$

Expressões, como $\log_e 4^2$, $\log_e (x+4)^2$ ou $\log_e \left(\sqrt{x+4}\right)^2$, possuem a mesma assinatura, *log_pwr*. Todas essas expressões poderiam ser classificadas como membros de uma certa classe de logaritmos. Todas elas possuem propriedades que são características da sua classe. Uma dessas propriedade é, por exemplo, o facto de todas elas serem transformáveis, por reescrita axiomática, numa expressão equivalente, com uma estrutura lógica do tipo $\text{prod}(\square, \log)$.

Cada assinatura define a hierarquia de classe a que pertence. Por exemplo, a assinatura *log_pwr* define a hierarquia

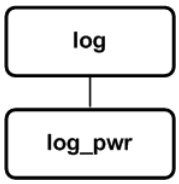



**Figura 2.4.8** ***A assinatura log_pwr***

Relativamente à base axiomática, em estudo, verificámos que apenas podem

ser consideradas as expressões com as seguintes assinaturas:

***Nível** 1:*

| | $op_1(a,b)$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $op_1$ | *sum* | *diff* | *prod* | *div* | *pwr* | *root* | *log* | *exp* | *der* |
| | $a+b$ | $a+b$ | $a\cdot b$ | $a/b$ | $b^a$ | $\sqrt[a]{b}$ | $\log_a b$ | $a^b$ | $D^a b$ |
| | | | | | | | | | |

*Expressões com estruturas* $\Xi$-$op_1$, *onde* $op_1 \in \mathcal{O}p$.

***Nível** 2:*

| | $op_1(op_2(a,b),\_)$ ou $op_1(\_,op_2(a,b))$ | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $op_2$ | *sum* | | *diff* | | *prod* | | *div* | | *pwr* | | *root* | | *log* | | *exp* | | *der* | |
| $op_1$ | | *E* | *D* | *E* | *D* | *E* | *D* | *E* | *D* | *E* | *D* | *E* | *D* | *E* | *D* | *E* | *D* | *E* | *D* |
| *sum* | $a+b$ | | | | | | | | | | | | | | | | | | |
| *diff* | $a-b$ | | | | | | | | | | | | | | | | | | |
| *prod* | $a\cdot b$ | | | | | | | | | | | | | | | | | | |
| *div* | $a/b$ | | | | | | | | | | | | | | | | | | |
| *pwr* | $b^a$ | | | | | | | | | | | | | | | | | | |
| *root* | $\sqrt[a]{b}$ | | | | | | | | | | | | | | | | | | |
| *log* | $\log_a b$ | | | | | | | | | | | | | | | | | | |
| *exp* | $a^b$ | | | | | | | | | | | | | | | | | | |
| *der* | $D^a b$ | | | | | | | | | | | | | | | | | | |

*Expressões com estruturas* $\Xi$-$op_1$_$op_2$, *onde* $op_1, op_2 \in \mathcal{O}p$. $op_1$ *está representado*

*em linha e* $op_2$ *em coluna. Cada coluna está dividida em duas: a da esquerda (E) representa estruturas* $op_1(op_2(a,b),\_)$ *e a da direita (D), estruturas* $op_1(\_,op_2(a,b))$.

***Nível** 3:*

| | $op_1(op_2(a,b),op_3(a,b))$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $op_1$ | *sum* | *diff* | | | | | | | |
| | $op_3$ | *sum* | *diff* | *prod* | *div* | *pwr* | *root* | *log* | *exp* | *der* |
| $op_2$ | | $a+b$ | $a-b$ | $a\cdot b$ | $a/b$ | $b^a$ | $\sqrt[a]{b}$ | $\log_a b$ | $a^b$ | $D^a b$ |
| *sum* | $a+b$ | | | | | | | | | |
| *diff* | $a-b$ | | | | | | | | | |
| *prod* | $a\cdot b$ | | | | | | | | | |
| *div* | $a/b$ | | | | | | | | | |
| *pwr* | $b^a$ | | | | | | | | | |
| *root* | $\sqrt[a]{b}$ | | | | | | | | | |
| *log* | $\log_a b$ | | | | | | | | | |
| *exp* | $a^b$ | | | | | | | | | |
| *der* | $D^a b$ | | | | | | | | | |

*Expressões com estruturas* $\Xi$-$op_1$_$op_2$_$op_3$, *onde* $op_1 \in \{sum, diff\}$ *e* $op_2, op_3 \in \mathcal{O}p$. $op_2$ *está representado em linha e* $op_3$ *em coluna.*

| | $op_1(op_2(a,b),op_3(a,b))$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $op_1$ | | | *prod* | *div* | | | | | |
| | $op_3$ | *sum* | *diff* | *prod* | *div* | *pwr* | *root* | *log* | *exp* | *der* |
| $op_2$ | | $a+b$ | $a-b$ | $a\cdot b$ | $a/b$ | $b^a$ | $\sqrt[a]{b}$ | $\log_a b$ | $a^b$ | $D^a b$ |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *sum* | $a+b$ | | | | | | | | | |
| *diff* | $a-b$ | | | | | | | | | |
| *prod* | $a \cdot b$ | | | | | | | | | |
| *div* | $a/b$ | | | | | | | | | |
| *pwr* | $b^a$ | | | | | | | | | |
| *root* | $\sqrt[a]{b}$ | | | | | | | | | |
| *log* | $\log_a b$ | | | | | | | | | |
| *exp* | $a^b$ | | | | | | | | | |
| *der* | $D^a b$ | | | | | | | | | |

*Expressões com estruturas* $\Xi\text{-}op_1\_op_2\_op_3$, *onde* $op_1 \in \{prod, div\}$ *e* $op_2, op_3 \in \mathcal{O}p$. $op_2$ *está representado em linha e* $op_3$ *em coluna.*

É assumido, nesta tese, que

> **o valor hipotético, representado pelo termo atómico $x$, não viola o domínio da função, representada hipoteticamente pelo termo funcional que o inclui como argumento.**

**Exemplo 2.4-4** ***Domínio de*** $\log_e 2x$

*Numa expressão, como* $\log_e 2x$, *o termo atómico* $x$ *é assumido como não violando o domínio da composição* $log \circ prod$, *representada pelo termo funcional com a assinatura* $log\_prod$. *Essa expressão seria, portanto, convertível numa outra com a assinatura* $sum\_log$, *ou seja, na expressão* $\log_e 2 + \log_e x$. *Porém, uma expressão, como* $\log_e 2(0)$, *seria convertida*

*para* $\log_e 0$ *e rejeitada por violação do domínio da função* $log$ .

---

A estrutura de uma expressão define aquilo que poderíamos designar por *classe de reescrita* dessa expressão. Seja $\rightarrow$ uma relação de equivalência, por reescrita axiomática, definida sobre o conjunto $\mathcal{T}$ .

**Definição 2.4-8** ***Classe de Reescrita***

> *A classe de reescrita de uma estrutura,* $\Xi$*-a , é o conjunto formado por todas as assinaturas da sua estrutura de classe,* $\mathcal{S}ig_a$ *e suas transformadas, por reescrita axiomática,* $\mathcal{T}Sig_a$*, ou seja, o conjunto*
>
> $$\mathcal{R}w_a = \mathcal{S}ig_a \cup \mathcal{T}Sig_a$$

**Exemplo 2.4-5** ***Classe de reescrita de*** $\Xi$*-log_pwr*

*A classe de reescrita de uma estrutura, como* $\Xi$*-log_pwr , é o conjunto* $\mathcal{R}w_{log\_pwr} = \{log, log\_pwr, prod, prod\_log\}$ *. A resolução de uma expressão como* $\log_e 4^2$*, com uma estrutura lógica do tipo* $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$*, envolveria duas classes de reescrita, ou seja, a classe*

$$\mathcal{R}w = \mathcal{R}w_{log\_pwr} \cup \mathcal{R}w_{pwr} =$$

$$= \{log, log\_pwr, prod, pwr, pwr\_pwr, prod\_log, pwr\}$$

---

Como veremos na Secção 4.3, as transformadas são obtidas, por reescrita axiomática:

- $log\_pwr \rightarrow prod\_log$ (por conversão),
- $log\_pwr \rightarrow log$ (por simplificação),
- $log \rightarrow log\_pwr$ (por factorização), e
- $pwr \rightarrow \square$ (por redução)

Por exemplo, a assinatura $log\_pwr$ é transformável, por reescrita axiomática, numa assinatura $prod\_log$.

**Definição 2.4-9** ***Classe de Equivalência***

> *Seja* $\rightarrow$ *uma relação de equivalência, definida sobre o conjunto* $\mathcal{T}$. *A classe de equivalência de uma expressão* $s \in \mathcal{T}$, *é o subconjunto* $\mathcal{E}q(s) \subseteq \mathcal{T}$, *formado por todas as expressões* $t \in \mathcal{T}$, *equivalentes a* $s$, *ou seja, o conjunto*
>
> $$\mathcal{E}q(s) = \{t \mid t \in \mathcal{T}, s \rightarrow t\}$$

As expressões obtidas, por reescrita axiomática, são membros de uma classe de equivalência. De facto, todas elas são formas equivalentes e, portanto, pertencentes a uma determinada classe de equivalência. Mas, nessa classe estão presentes também todas as formas equivalentes dessas expressões que não podem ser obtidas, por reescrita axiomática. Por exemplo, não é possível obter a fracção $\frac{2}{4}$ por uma reescrita axiomática da fracção $\frac{5}{10}$.

Assim, consideramos que

> **a resolução simbólica de uma expressão, por reescrita axiomática, é simplesmente um subconjunto da sua classe de equivalência.**

De facto, nem todas as assinaturas se podem obter, por reescrita axiomática. Uma reescrita axiomática só produz expressões cujas assinaturas fazem parte da sua classe de reescrita. Por exemplo, assinaturas como *log_sum* ou *log_log* não fazem parte de nenhuma classe de reescrita. De facto, relativamente aos axiomas e teoremas que são utilizados no ensino secundário, não conseguimos encontrar nenhum com esse tipo de assinatura. Todos os axiomas incluídos nesta tese estão listados no Apêndice B e lidam apenas com o tipo de assinaturas que foram apresentadas anteriormente e que são as únicas que são tratadas nesta tese.

**Exemplo 2.4-6** ***Classe de reescrita de*** $\log_e \log_2 4^2$

*Expressões como* $\log_e(x+4)$ *ou* $\log_e \log_2 4^2$*, não são transformáveis, através dessas assinaturas. A expressão* $\log_e \log_2 4^2$ *é, porém, transformável através da sua componente,* $\log_2 4^2$*, cuja assinatura é* $log\_pwr$*. A sua classe de reescrita seria o conjunto*

$$\mathcal{R}w = \{sum, sum\_log\_log, prod, prod\_log,$$

$$pwr, pwr\_pwr, log, log\_prod, log\_pwr\}$$

---

As classes de reescrita estão organizadas, hierarquicamente, por assinaturas. Seja $\prec$ uma ordem parcial estrita e $(\mathcal{O}p, \prec)$ um conjunto parcialmente ordenado. A ordenação das assinaturas é feita com base numa ordem lexicográfica, induzida por $(\mathcal{O}p, \prec)$. Na Figura 2.4.9, está representada a estrutura de classes para o operador *log* (logaritmos).

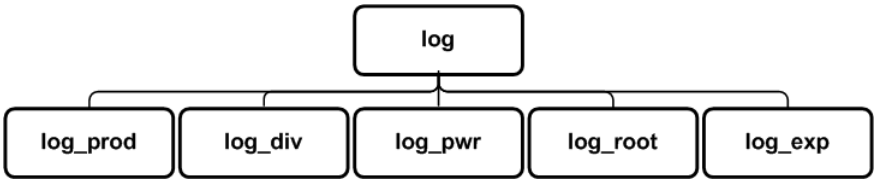



**Figura 2.4.9** *Estrutura da classe log*

Como se pode observar, nessa estrutura, não estão incluídas assinaturas como *log_sum* ou *log_log* . No caso do operador logaritmo, só foram necessários definir assinaturas de nível 1 e 2, ou seja, assinaturas do tipo *op* e *op_op* .

A base axiomática utilizada nesta tese necessitou apenas de três (3) níveis hierárquicos para definir todos os axiomas e teoremas. O axiomas e teoremas utilizados, nesta tese, estão listados no Apêndice B. Os três níveis estruturais de uma estrutura triangular ( N1 , N2 e N3 ) foram suficientes para definir os três (3) níveis hierárquicos da base axiomática.

**Definição 2.4-10** ***Base Axiomática***

> *A base axiomática de uma assinatura, denotada por* $\mathcal{B}_{Sig}$ *, é o conjunto de todas as regras de reescrita cujas assinaturas são membros da sua classe de reescrita, ou seja,*
>
> $$\mathcal{B}_{sig} = \left\{ l_i \to r_j \mid i \neq j \wedge i, j \in \mathcal{R}w_{sig} \right\}$$
>
> *onde* $\mathcal{R}w_{sig}$ *é a classe de reescrita da assinatura* $Sig$ *.*

Como veremos na Secção 4.3.1, a base axiomática para cada operador é constituída por regras de reescrita com esse tipo de estrutura. A base axiomática é completa se todas as classes de reescrita possuírem regras de reescrita para todos os seus membros.

Como veremos na Secção 4.4, a resolução simbólica de expressões é

baseada nesse tipo de estrutura. As expressões são caracterizadas com base na estrutura triangular das suas componentes principais. Nesta tese, as expressões são resolvidas, por reescrita axiomática, componente a componente. Expressões mais complicadas como, por exemplo, os polinómios, não foram incluídos nesta tese, pois para a sua caracterização são necessárias mais do que uma componente.

Como veremos também, as estratégias de resolução estão intimamente relacionadas com esse tipo de estrutura. Por exemplo, expressões com assinaturas do tipo *sum_log_log* só pode ser tratadas, axiomaticamente, como membros dessa classe ou da classe *sum*, donde derivam. Neste caso, a classe *sum_log_log* deriva, directamente, da classe *sum*, e não de uma hipotética classe *sum_log* . A classe *sum_log* não faz parte da base axiomática. De facto, no âmbito dos axiomas e teoremas aplicados no ensino secundário, não existe nenhum que possua esse tipo de assinatura.

**Exemplo 2.4-7** ***Assinatura de*** $\log_e x^2 + 3$

*Expressões como* $\log_e x^2 + 3$ *só podem ser tratadas, axiomaticamente, como somas, ou seja, resolvidas como*

$$\log_e x^2 + 3 \to 2\log_e x + 3$$

---

Porém, para expressões como $\log_e x^2 + \log_e x^3$, existem axiomas com essa assinatura, isto é, axiomas com a assinatura *sum_log_log* . Essas expressões podem ser tratadas, axiomaticamente, como somas ou somas de logaritmos.

Expressões como 2, 4 ou $x$ possuem estruturas do tipo Ξ-*none* e, portanto, são expressões sem assinatura. Essas expressões são consideradas membros de uma

classe hipotética *none*. E, como tal, são irredutíveis. Mas, como veremos na Secção 4.3.1.4, em determinados contextos, certos números podem ser factorizados. A factorização de números inteiros é a única transformação lógica possível para alguns membros desta classe.

**Exemplo 2.4-8** ***Factorização de*** $\log_e 16$

*Numa expressão como* $\log_e 16$*, a constante 16 seria factorizável. Mas já o mesmo não sucederia se essa constante aparecesse descontextualizada.*

---

Como veremos na Secção 4.3.1.4, a factorização de números inteiros é uma forma de reescrita que designamos por contextualizada.

O Prolog[10] foi a linguagem que escolhemos para implementar esse tipo de estrutura. Nessa linguagem, as expressões lógicas são representadas por termos simples ou compostos. Os lados esquerdos de fórmulas matemáticas estão representados por expressões lógicas com variáveis livres. As bases axiomáticas estão organizadas, hierarquicamente, com base na assinatura do lado esquerdo dessas fórmulas.

**Exemplo 2.4-9** ***Representação lógica de*** $\log_s u^t \to t \log_s u$

*Uma fórmula como* $\log_s u^t \to t \log_s u$*, é representada por um conjunto de regras de reescrita condicionais do tipo* $\log(s, \mathrm{pwr}(t,u)) \to \mathrm{prod}(t, \log(s,u))$*, com assinaturas* `log_pwr`*, e organizadas, hierarquicamente, dentro da classe* `log`*. Os termos* `s`*,* `t` *e* `u` *são variáveis lógicas.*

---

[10] Para esta tese, foi utilizada a versão 3.9.1 da linguagem SICStus™ Prolog.

A aplicabilidade de uma fórmula é apenas determinada pela sua assinatura e pelas condições que são impostas às suas variáveis.

**Exemplo 2.4-10** ***Reescrita de*** $\log_e(2\sqrt{x}+1)^2$

*Uma expressão como* $\log_e(2\sqrt{x}+1)^2$, *com uma assinatura* $log\_pwr$, *poderia ser reescrita, axiomaticamente, como* $2\log_e(2\sqrt{x}+1)$, *aplicando uma das regras de reescrita, com essa assinatura, que satisfaça aos seus requisitos operacionais.*

---

A aplicação de uma fórmula depende, naturalmente, da estratégia escolhida. Como veremos na Secção 4.4.2, a aplicação de uma estratégia pode levar a expressões mais complexas.

Consideramos que

> **o importante é assegurar que a aplicação de uma regra reduza ou forme novos *redexes*[11] e que, na base axiomática, existam regras para reduzir esses *redexes*.**

De facto, todas as regras, na base axiomática, cumprem com esse objectivo. Isto é, na base axiomática só existem regras redutoras ou potencialmente redutoras.

**Exemplo 2.4-11** ***Reescrita de*** $\log_{10} 200$

*A expressão* $\log_{10} 200$ *é reescrita como* $\log_{10} 2+2$, *porque a factorização do*

[11] Uma abreviação da expressão inglesa "***red****ucible* ***ex****pression*" [VAL00]. O termo é referenciado em muitos artigos que lidam com o cálculo lambda e a redução-$\beta$. Não sabemos, porém, donde o termo proveio ou quem o cunhou.

*seu argumento,* $2(10^2)$*, potencia a formação de dois novos redexes*

$$\log_{10} 200 \rightarrow \underbrace{\log_{10} 2(10^2)}_{redex} \rightarrow \log_{10} 2 + \underbrace{\log_{10} 10^2}_{redex} \rightarrow \log_{10} 2 + 2$$

Como veremos na Secção 4.3, a factorização de inteiros é um tipo de reescrita contextualizada, que contribui para a formação de *redexes*.

### *2.4.2 O Conceito de Poder Redutor*

A estrutura Ξ-*none* é considerada a mais simples das cinco (5) estruturas triangulares básicas. Nesse tipo de estruturas, estão apenas presentes termos atómicos. Estruturas do tipo Ξ-*none* são estruturas irredutíveis.

Consideramos que

> **a presença de termos atómicos em qualquer estrutura não altera o grau de complexidade dessa estrutura.**

De facto, por serem irredutíveis, as estruturas Ξ-*none* não consumem nem produzem novas estruturas.

Consideramos também que

> **o grau de complexidade de uma expressão depende, essencialmente, da estrutura triangular da sua componente principal.**

De facto, cada componente possui a sua própria estrutura triangular e cada componente pode ser considerada como sendo a componente principal de uma expressão ou subexpressão. O número de componentes presentes numa expressão depende apenas do nível estrutural de cada componente. De facto, o nível de uma estrutura determina o nível de componentes presentes nessa estrutura. Por exemplo, uma estrutura do nível 0 não possui nenhuma (0) componente, enquanto que uma

do nível 3 possui três (3) componentes.

**Exemplo 2.4-12** ***Componentes de*** $\log_e \sqrt{4^3}$

*Uma expressão, como* $\log_e \sqrt{4^3}$, *com uma estrutura do tipo* $[\Xi\text{-}log\_root][\Xi\text{-}root\_pwr][\Xi\text{-}pwr]$, *possui três (3) componentes. A terceira, porém, faz parte apenas da sua subexpressão* $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$.

---

A Figura 2.4.10 ilustra esse facto. A expressão logaritmica possui apenas uma (1) componente, a expressão raiz, embora esta última possua também uma (1) componente, a expressão potência.

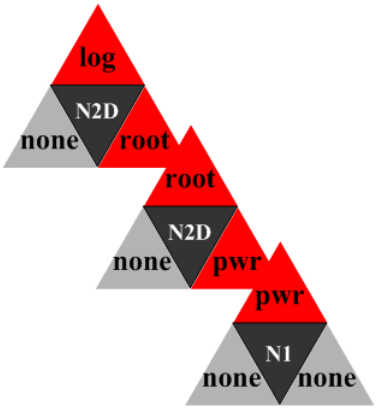



**Figura 2.4.10** ***As componentes de um expressão***

As estruturas mais complexas são aquelas com um ou mais termos funcionais. Como vimos atrás, numa estrutura triangular básica podem estar presentes, no máximo, três termos funcionais. As estruturas mais complexas são, por conseguinte, as estruturas de nível 3, ou seja, estruturas de tipo $\Xi\text{-}op\_op\_op$. Numa estrutura desse tipo estão sempre presentes 3 componentes. O nível estrutural de uma expressão é determinado apenas pelo nível estrutural da sua componente

principal. O número de componentes visíveis em cada expressão, depende apenas do nível estrutural da sua componente principal. Por exemplo, numa estrutura de nível 3, o número de componentes visíveis é sempre 3, independentemente do nível estrutural das outras duas. O mesmo se poderia dizer dos outros 3 níveis estruturais ( N0 , N1 e N2 ). Numa estrutura de nível 2, esse número baixaria para 2, e numa de nível 0, para uma ausência total de componentes.

Assim, consideramos que

> **o grau de complexidade de uma estrutura triangular depende do seu nível estrutural.**

Por exemplo, uma estrutura do tipo $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr\_root][\Xi\text{-}root]$ teria o mesmo grau de complexidade que uma do tipo $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$, pois em ambas estariam visíveis apenas duas componentes: a principal $[\Xi\text{-}log\_pwr]$ e a que está à profundidade 1, isto é, $[\Xi\text{-}pwr\_root]$ ou $[\Xi\text{-}pwr]$. De facto, ambas são estruturas de nível 2, com a mesma assinatura $log\_pwr$.

O que é que sucederia, porém, se essas estruturas fossem do mesmo nível, mas com assinaturas diferentes? Por exemplo, uma expressão como $\log_e 4^2$, com uma estrutura do tipo $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$, e uma expressão como $4\log_e 2$, com uma estrutura do tipo $[\Xi\text{-}prod\_log][\Xi\text{-}log]$. Ambas são de nível 2, mas com assinaturas diferentes, $log\_pwr$ e $prod\_log$. Com base no seu nível estrutural, ambas deveriam ser consideradas como tendo o mesmo grau de complexidade. Só que, como veremos a seguir, a estrutura $\Xi\text{-}prod\_log$ é, de facto, a mais simples das duas, pois representa, matematicamente, a forma mais simples de se efectuar o cálculo desse logaritmo. A assinatura $prod\_log$ denota, de facto, uma simples contagem de elementos com a assinatura $log$.

Consideramos que

**o grau de complexidade de uma estrutura depende da sua assinatura.**

De facto, a assinatura de uma expressão define o seu nível estrutural, ou seja, o número de componentes presentes nessa estrutura a essa profundidade.

Note que só é importante comparar estruturas que sejam equivalentes. Do ponto de vista de uma reescrita axiomática, essa comparação é necessária para que se possa saber qual das estruturas é a mais simples. Essa decisão é tomada ao nível da reescrita e com base no grau de complexidade das estruturas envolvidas. As regras de reescrita são portanto quem decide sobre esse aspecto da resolução, condicionando a sua aplicação à redução ou formação de *redexes*. É através desse condicionamento que é controlado o poder redutor dessas regras. Tanto o poder redutor como o grau de complexidade são definidos a seguir.

Como vimos na Secção 2.4.1, a classe de equivalência de uma expressão é o conjunto de todas as formas equivalentes dessa expressão. Mas nem todas essas formas se podem obter por reescrita axiomática.

**Exemplo 2.4-13** ***Transformação não axiomática*** $\log_5 5^3 \rightarrow_? 6-3$

*Não é possível transformar* $\log_5 5^3$ *em* $6-3$*, por reescrita axiomática. Qual seria a regra a aplicar?*

---

Para que duas estruturas triangulares sejam consideradas equivalentes, por reescrita axiomática, é necessário algo mais que o simples facto de serem apenas equivalentes. Duas expressões podem ser equivalentes e não o serem por reescrita axiomática. Para que o sejam, é necessário que, pelo menos, uma delas seja transformável na outra, por reescrita axiomática, ou seja,

**Definição 2.4-11** ***Estruturas Equivalentes***

> *Diz-se que duas estruturas triangulares,* $\Xi$-$a$ *e* $\Xi$-$b$*, são equivalentes, e escreve-se,* $\Xi\text{-}a \leftrightarrow \Xi\text{-}b$*, se e só se existir, pelo menos, uma regra de reescrita,* $l_a \to r_b \in \mathcal{B}$*, com* $a \neq b$*, que as transforme uma na outra.*

A resolução simbólica de expressões não visa apenas obter formas equivalentes dessas expressões. Visa, essencialmente, a simplificação dessas expressões, ou seja, a aplicação de um número limitado de regras de reescrita que reduzam, ou contribuam para a redução do grau de complexidade dessas estruturas. Essa capacidade de redução é aquilo que designamos, nesta tese, por poder redutor. O poder redutor de uma regra é determinado pelo grau de complexidade do seu lado direito. Quanto mais simples for essa estrutura, tanto maior será o seu poder redutor.

**Definição 2.4-12** ***Poder Redutor***

> *Sejam,* $\Xi$-$a$ *e* $\Xi$-$b$*, com* $a \neq b$*, duas estruturas equivalentes e* $\triangleright$ *uma ordem parcial estrita ou poder redutor da estrutura. Seja* $\triangleleft$ *a inversa de* $\triangleright$ *e* $\mathcal{N}(\Xi\text{-}Sig)$*, o nível da estrutura. Então*
>
> *A. Diz-se que* $\Xi$-$a$ *é redutora, ou que possui um elevado poder redutor, e escreve-se* $\Xi\text{-}a \triangleright \Xi\text{-}b$*, se*
>
> *i)* $\mathcal{N}(\Xi\text{-}a) > \mathcal{N}(\Xi\text{-}b)$*, ou*
>
> *ii)* $\mathcal{N}(\Xi\text{-}a) = \mathcal{N}(\Xi\text{-}b)$ *e* $\Xi$-$b$ *for uma estrutura mais simples*
>
> *B. Diz-se que* $\Xi$-$a$ *é potencialmente redutora, ou que possui um potencial poder redutor, e escreve-se* $\Xi\text{-}a \triangleleft \Xi\text{-}b$*, se*
>
> *iii)* $\mathcal{N}(\Xi\text{-}a) < \mathcal{N}(\Xi\text{-}b)$ *e* $\Xi$-$b$ *for redutora*
>
> *C. Caso contrário, diz-se que* $\Xi$-$a$ *é terminal, não redutora ou que possui poder redutor nulo.*

No caso das duas estruturas serem do mesmo nível, importa ainda considerar que

> **a mais simples é aquela que representar um *produto* por uma constante, ou uma *potência*.**

De facto, estruturas do tipo prod($k$, _) ou pwr($k$, _) representam o que normalmente poderíamos designar por *contagens* de *objectos* idênticos presentes nessas estruturas. Uma estrutura triangular que represente uma *listagem* desses objectos, teria certamente que possuir mais componentes.

Note que este facto foi também constatado em casos onde as estruturas são de níveis diferentes, mas uma das estruturas é uma constante racional. Nesses casos, o agravamento do grau de complexidade é compensado pelo facto de uma das componentes ser uma constante, irredutível ou potencialmente redutível. Por exemplo, a fracção $\frac{3}{4}$ é uma constante racional irredutível, enquanto que a fracção $\frac{5}{10}$ é potencialmente redutível.

Em todos os outros casos, a regra mantém-se, isto é, a expressão mais simples é aquela que o nível mais baixo (isto é, com menos componentes).

As seguintes transformações assim o evidenciam

- $\sum_k x \to kx$ ou $\sum_k -x \to -kx$ (distributividade)

  Note que para $k = 2$ tem-se, de facto, duas estruturas do mesmo nível. Por indução, esta mesma transformação seria aplicável a somatórios[12] de qualquer outro tipo de "*objecto*", incluindo produtórios de outros

[12] O somatório é entendido, aqui, como somas binárias de somas binárias.

"*objectos*". Nesta tese, foram incluídos somatórios de todas as estruturas $\Xi$-*Sig* , e produtório de estruturas $\Xi$-*none* , $\Xi$-*prod* , $\Xi$-*div* , $\Xi$-*pwr* , $\Xi$-*root* e $\Xi$-*exp* .

- $kx \pm k \to k(x \pm 1)$ (distributividade)

  Note que, seja qual for $k$ tem-se, de facto, duas estruturas do mesmo nível.

- $\prod_k x \to x^k$ ou $\prod_k -x \to (-x)^k$ (equivalência de produtos e potências)

  Note que para $k = 2$ tem-se, de facto, duas estruturas do mesmo nível. Por indução, esta mesma transformação seria aplicável a produtórios[13] de qualquer outro tipo de "*objecto*" . Nesta tese, só foram incluídos produtórios de estruturas $\Xi$-*none* , $\Xi$-*prod* , $\Xi$-*div* , $\Xi$-*pwr* , $\Xi$-*root* e $\Xi$-*exp* .

- $\log_a \prod_k x \to \log_a x^k \to k \log_a x$ (equivalência entre logaritmos de produtos e produtos de logaritmos)

  Note que para $k = 2$ tem-se, de facto, duas estruturas do mesmo nível. Por indução, esta mesma transformação seria aplicável a produtórios de qualquer outro tipo de "*objecto*" . Nesta tese, só foram incluídos produtórios de estruturas $\Xi$-*none* , $\Xi$-*prod* , $\Xi$-*div* , $\Xi$-*pwr* , $\Xi$-*root* e $\Xi$-*exp* ).

Consideramos ainda que

[13] O produtório é entendido, aqui, como produtos binários de produtos binários.

> **na base axiomática, só podem existir regras *redutoras* ou *potencialmente redutoras*.**

De facto, regras que não sejam

- Redutoras, isto é, produtoras de estruturas mais simples, ou
- Potencialmente redutoras, isto é, produtoras de estruturas mais complexas mas com maior poder redutor,

só podem produzir estruturas com o mesmo grau de complexidade ou superior, mas com um poder redutor inferior ou igual. O resultado seria, portanto, um agravamento progressivo do grau de complexidade dessas estruturas. Regras desse tipo não contribuiriam, certamente, para a simplificação dessas expressões.

**Exemplo 2.4-14** ***Assinatura*** *log_pwr* ***e*** *prod_log*

> *No caso das regras, com assinaturas* $\mathsf{log\_pwr}$ *e* $\mathsf{prod\_log}$*, a que possui a assinatura* $\mathsf{log\_pwr}$ *é redutora. A outra é potencialmente redutora.*

---

A regra $\log(s,\mathrm{pwr}(t,r)) \to \mathrm{prod}(t,\log(s,r))$ seria, portanto, mais redutora que a sua inversa. Como se sabe, essa fórmula traduz uma relação fundamental entre logaritmos e potências. Numa resolução simbólica, é importante saber qual o sentido mais redutora de uma fórmula.

**Definição 2.4-13** ***O Grau de Complexidade***

> *Sejam,* $\Xi$*-a e* $\Xi$*-b, com* $a \neq b$*, duas estruturas equivalentes,* $\mathcal{N}(\Xi\text{-}\mathcal{S}ig)$*, o nível de uma estrutura e* $\mathcal{N}\mathcal{R}edex(\Xi\text{-}\mathcal{S}ig)$*, o número de redexes numa estrutura. Diz-se que a estrutura* $\Xi$*-a é mais complexa, ou que possui um maior grau de complexidade, e escreve-*

> *se* $\Xi\text{-}a > \Xi\text{-}b$ *, se e só se*
>
> *i)* $\mathcal{N}(\Xi\text{-}a) \geq \mathcal{N}(\Xi\text{-b})$ *, e*
>
> *ii)* $\mathcal{NR}edex(\Xi\text{-}a) \geq \mathcal{NR}edex(\Xi\text{-}b)$

Consideramos que

> **o grau de complexidade de uma estrutura deve variar no mesmo sentido que o seu poder redutor.**

De facto,

- Numa redução, o grau de complexidade de uma estrutura diminui assim como o seu poder redutor e, portanto, ambos variam no mesmo sentido.

- Numa conversão, porém, o grau de complexidade de uma estrutura pode aumentar ou diminuir.

    - No caso do grau de complexidade diminuir, não há formação de *redexes* e, portanto, o poder redutor diminui também. A diminuição do poder redutor é traduzido pelo número de *redexes* consumidos (ou seja, 1).

    - No caso do grau de complexidade aumentar, há formação de *redexes* e, portanto, o poder redutor tem de aumentar para compensar esse agravamento. O aumento do poder redutor é traduzido pelo número de *redexes* formados (ou seja, 1 ou 2).

- Caso contrário, se o poder redutor variar em sentido oposto, então o resultado seria um agravamento progressivo do grau de complexidade dessa estrutura.

**Exemplo 2.4-15** ***Expressões equivalentes*** $\log_e 2^4$ ***e*** $4\log_e 2$

*As expressões* $\log_e 2^4$ *e* $4\log_e 2$ *são duas estruturas equivalentes, do mesmo nível, mas com poderes redutores diferentes. A primeira é a que possui o poder redutor mais elevado e, portanto, a que é considerada a mais complexa. A outra é apenas potencialmente redutora.*

---

Porém, neste caso o *redex* formado é apenas redutível a si próprio. Casos como este, são detectados pelo princípio de exclusão e, portanto, evitados.

Consideramos que,

> **numa reescrita axiomática, é importante que se eliminem ou se formem novos *redexes*.**

De facto, como vimos atrás,

- A eliminação de *redexes* diminui o grau de complexidade da estrutura e, portanto, torna-a mais simples.

- A formação de novos *redexes*, compensa o agravamento do grau de complexidade, possibilitando reduções a posteriori.

Como veremos, na Secção 4.3.1, o poder redutor de uma fórmula é determinado pelo poder redutor do seu lado esquerdo. Por exemplo, o poder redutor de uma fórmula, como $\log(s,\text{pwr}(t,u)) \rightarrow \text{prod}(t,\log(s,u))$, é determinado pela estrutura $\log(\square,\text{pwr})$ do seu lado esquerdo. As fórmulas estão dispostas por ordem decrescente do seu poder redutor. Como no Prolog, essa ordem é, estritamente, sequencial , as regras estão escritas nessa ordem.

## 2.5 A Transformação Lógica de Expressões

A transformação lógica de expressões matemáticas envolve a reescrita

axiomática dessas expressões, ou seja, a sua transformação através da aplicação sistemática de um certo número de regras de reescrita (axiomas e teoremas elementares). Para se efectuar uma reescrita axiomática é necessário uma base axiomática relativamente completa.

Os axiomas e teoremas elementares, que constituem a base axiomática utilizada nesta tese, estão listadas no Apêndice B.

**Definição 2.5-1** ***Transformação Lógica***

> *Uma transformação lógica é um processo de reescrita de expressões, por via axiomática, representada por* $s \rightarrow_{\Gamma} t \equiv \Gamma(s)$, *que conduz à substituição de uma expressão* $s$ *por uma outra* $t$, *que lhe é equivalente. A expressão* $t$ *é designada por transformada lógica de* $s$.

De acordo com esta definição, a expressão $5$ seria a transformada lógica, por cálculo, da expressão $2+3$, ou seja, o resultado da transformação lógica, $\Gamma(2+3)$. A questão que se põe aqui é se essa é a única transformação possível para a expressão $2+3$? Neste caso, não restam quaisquer dúvidas, pois trata-se, efectivamente, de um simples cálculo numérico, ou seja, de uma transformação do tipo $2+3 \rightarrow 5 \equiv \Gamma(2+3)$. Essa transformação faria todo o sentido, pois aplicando-a, estaríamos, de facto, a *eliminar* operadores e a tornar, portanto, a expressão mais simples.

Mas segundo essa definição, também seria possível transformar a expressão $5$ numa outra que lhe fosse equivalente. Só que, neste caso, a transformação não faria sentido, pois aplicando-a, estaríamos a *adicionar* operadores, ou seja, a tornar a expressão mais complexa. De facto, a expressão $5$ já se encontra representada na sua forma irredutível. Essa é a forma que [BAA98, DER01] designam por normal.

**Definição 2.5-2** ***Forma Normal***

> *A forma normal de uma expressão, $s \in \mathcal{T}$ , é uma forma irredutível dessa expressão, denotada por $\mathcal{N}orm(s)$, isto é, uma forma que não contém nenhum redex [BAA98, DER01, VAL00]. Se essa forma for única, então essa forma é a mais simples.*

A forma normal de uma expressão não tem que ser necessariamente a sua forma mais simples. A forma normal é apenas uma forma irredutível dessa expressão. A sua forma mais simples é, porém, por definição, irredutível e, por conseguinte, normal. Como veremos, mais adiante, as formas normais não contribuem para a simplificação de expressões matemáticas, pois, por definição, não contêm *redexes* nem potenciam a formação de *redexes*. As formas normais contribuem, apenas, para a terminação de cadeias de transformações. Por exemplo, a transformação $\Gamma(5)$, não contribui para a simplificação da expressão 5. Mas como uma forma irredutível, a expressão 5 contribui para a terminação da resolução de $2+3$.

Consideramos que

> **numa transformação lógica, nem sempre a forma obtida é mais simples que aquela donde derivou.**

De facto, o resultado de uma transformação lógica pode ser uma expressão com um grau de complexidade mais elevado. Mas como vimos atrás, esse agravamento é compensado por um poder redutor mais elevado.

Podemos dizer que

- Quando o resultado é uma expressão mais simples, a transformação é uma *redução* (a forma obtida é normal) ou uma *conversão redutora* (a forma obtida é ainda redutível ).

- Quando a forma obtida é mais complexa que aquela donde derivou, a transformação é uma *conversão potencialmente redutora* (a forma é potencialmente redutível).

Note que, numa transformação lógica, a forma obtida não pode ser mais complexa e normal, pois isso violaria o objectivo principal de uma transformação lógica que é reduzir ou formar novos *redexes*. De facto, na base axiomática não existem regras desse tipo.

Numa conversão redutora, o resultado da transformação é apenas, estruturalmente, mais simples. Existem ainda *redexes* por resolver.

**Exemplo 2.5-1** ***Conversão redutora*** $\frac{\mathrm{d}}{\mathrm{dx}}(4+1) \rightarrow \frac{\mathrm{d}}{\mathrm{dx}}5$

*A transformação* $\frac{\mathrm{d}}{\mathrm{dx}}(4+1) \rightarrow \frac{\mathrm{d}}{\mathrm{dx}}5$ *é uma conversão redutora, pois a transformada é mais simples mas ainda redutível.*

---

Numa conversão potencialmente redutora, o resultado é, estruturalmente, mais complexo, mas possui um ou mais *redexes* por resolver.

**Exemplo 2.5-2** ***Conversão potencialmente redutora*** $\frac{\mathrm{d}}{\mathrm{dx}}(x+1) \rightarrow \frac{\mathrm{d}}{\mathrm{dx}}x + \frac{\mathrm{d}}{\mathrm{dx}}1$

*A transformação* $\frac{\mathrm{d}}{\mathrm{dx}}(x+1) \rightarrow \frac{\mathrm{d}}{\mathrm{dx}}x + \frac{\mathrm{d}}{\mathrm{dx}}1$ *é uma conversão desse tipo. A expressão* $\frac{\mathrm{d}}{\mathrm{dx}}x + \frac{\mathrm{d}}{\mathrm{dx}}1$ *é mais complexa mas potencialmente redutível.*

---

Qualquer expressão, com um ou mais *redexes*. é redutível. Assim, uma expressão, redutora ou potencialmente redutora é redutível.

Podemos então dizer que existem três tipos de transformações lógicas.

**Definição 2.5-3** ***Redução***

> *Uma redução é uma transformação lógica,* $\Gamma_R$*, cujo resultado é sempre uma forma normal (irredutível), isto é se* $s \in \mathcal{T}$*, então* $\Gamma_R(s) = \mathcal{N}orm(s)$.

Uma redução é uma transformação que conduz sempre a uma forma normal dessa expressão. A transformada de uma redução é sempre uma expressão mais simples. Se essa forma for a única, então essa forma é também a forma mais simples dessa expressão.

**Exemplo 2.5-3** ***Redução*** $x+0 \rightarrow x$

*A transformação* $x+0 \rightarrow x$ *é uma redução.*

---


**Figura 2.5.1** ***Redução***

**Definição 2.5-4** ***Conversão***

> *Uma conversão é uma transformação lógica,* $\Gamma_T$*, cujo resultado pode ser uma forma normal, ou uma forma, redutora ou potencialmente redutora, isto é, se* $s \in \mathcal{T}$ *então* $\Gamma_T(s) = \mathcal{N}orm(s)$

*ou* $\Gamma_T(s) = \mathcal{R}edex(s)$.

Note que uma conversão nem sempre produz uma forma simplificada da expressão.

**Exemplo 2.5-4** ***Conversão*** $\log_e 4^2 \rightarrow 2\log_e 4$

*A transformação lógica,* $\log_e 4^2 \rightarrow 2\log_e 4$*, é uma conversão.*

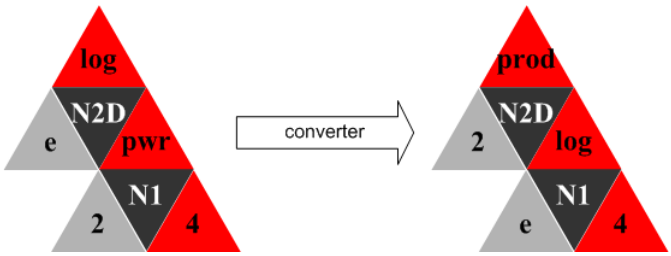



**Figura 2.5.2** ***Conversão***

Tanto as reduções como as conversões são transformações lógicas que resultam da aplicação directa de fórmulas matemáticas, com estruturas triangulares bem definidas. Essas estruturas foram abordadas, atrás, na Secção 2.4.

Quando a aplicação é feita de forma indirecta, temos aquilo que designamos por simplificação, ou seja, a aplicação indirecta de fórmulas matemáticas a determinadas posições da estrutura.

**Definição 2.5-5** ***Simplificação***

*Uma simplificação é uma transformação lógica,* $\Gamma_S$*, cujo resultado é a aplicação indirecta de reduções, conversões ou a aplicação*

*recursiva de outras simplificações*[14]*, isto é, se* $s,t \in \mathcal{T}$ *e* $t[s]$ *for um subtermo de* $t$*, então*

$$\Gamma_S(t[s]) = t\left[\Gamma_R(s)\right],\ \Gamma_S(t[s]) = t\left[\Gamma_T(s)\right] \text{ ou } \Gamma_S(t[s]) = t\left[\Gamma_S(s)\right].$$

A aplicação indirecta de fórmulas matemáticas é feita de forma recursiva, utilizando uma estratégia de procura, em profundidade, da esquerda para a direita.

**Exemplo 2.5-5** ***Simplificação*** $\log_e 4^2 \rightarrow \log_e 16$

*A transformação lógica,* $\log_e 4^2 \rightarrow \log_e 16$*, é uma simplificação, pois resulta da aplicação indirecta de uma regra de cálculo a uma das suas subestruturas, nomeadamente a expressão* $4^2$*.*

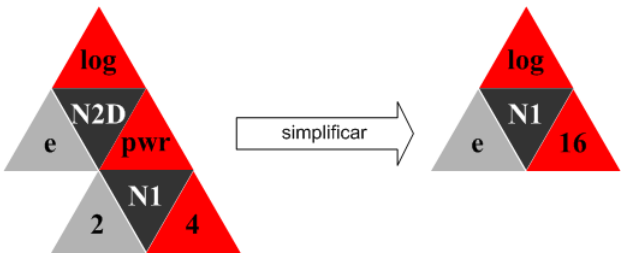



**Figura 2.5.3** ***Simplificação***

No caso de uma expressão, como $\log_e 4^2$, seriam necessárias várias transformações lógicas, para a resolver essa expressão, completamente, de forma simbólica. Uma possível solução seria, por exemplo, a seguinte sequência de transformações lógicas

$$\log_e 4^2 \rightarrow_{\Gamma_1} \log_e 16 \rightarrow_{\Gamma_2} \log_e 2^4 \rightarrow_{\Gamma_3} 4\log_e 2$$

[14] As simplificações são, por natureza, transformações lógicas recursivas.

Mas, como se pode observar, nem todas as transformações lógicas conduzem, necessariamente, a formas simplificadas dessa expressão. Nessa cadeia, a transformação lógica, $\Gamma_2$, é uma simplificação que conduz a uma expressão mais complexa, $\log_e 2^4$, mas potencialmente redutora. Todas elas, porém, contribuem para a eventual normalização ou simplificação dessa expressão. A expressão $4\log_e 2$ é, neste caso, a mais simples. A resolução simbólica da expressão $\log_e 4^2$ será abordada, com um pouco mais detalhe, no Capítulo 4.

É preciso notar também que nem sempre a forma normal de uma expressão é a sua forma mais simples. Isso só acontece se a base axiomática for convergente [BAA98, DER01]. Neste caso, a forma normal é única [BAA98]. Como vimos atrás, a forma normal de uma expressão é simplesmente uma forma irredutível dessa expressão. A resolução simbólica de expressões é uma sequência lógica de transformações lógicas que depende fortemente da completude da sua base axiomática para garantir a convergência do seu sistema de reescrita.

Consideramos que

> **a resolução simbólica de expressões envolve apenas um número limitado de reduções, conversões e simplificações.**

De facto, como vimos atrás, as transformações lógicas, envolvem apenas transformações lógicas que sejam redutoras ou potencialmente redutoras e que possam ser aplicadas a qualquer parte de uma estrutura triangular. É certo que não podemos garantir a completude da base axiomática que foi utilizada nesta tese, embora tenhamos construído uma relativamente completa, ao nível do secundário, cujos axiomas e teoremas estão listados no Apêndice B. Essa base está listada no Apêndice B. O Apêndice C apresenta um conjunto de exemplos que pretende ser representativo da generalidade dos programas do secundário. Para todos os exemplos, os testes efectuados demonstram a completude da base axiomática apresentada nesta tese.

**Definição 2.5-6** ***Resolução Simbólica***

> *A resolução simbólica de uma expressão* $s \in \mathcal{T}$ *é um conjunto ordenado de transformações lógicas,* $\mathcal{R}es = \{\Gamma_0(s), \ldots, \Gamma_n(s)\}$, *que conduzem a uma forma irredutível dessa expressão* $t \in \mathcal{T}$, *ou seja, uma sequência lógica do tipo*
>
> $$s \to \Gamma_1(s) \to \cdots \to \Gamma_i(\ldots(\Gamma_1(s))) \to \cdots \to \Gamma_{n-1}(\ldots(\Gamma_1(s))) \to t$$
>
> *onde*
>
> $$s \equiv \Gamma_0(s) \;\; e \;\; t \equiv \Gamma_n(\ldots(\Gamma_1(s)))$$

Uma resolução simbólica pode ter várias soluções. Todas elas devem conduzir, no entanto, ao mesmo resultado. A cada uma dessas soluções demos o nome de solução simbólica.

## 2.6 Conclusões

Neste capítulo foi apresentado o objecto do nosso estudo, ou seja, o desenvolvimento de estruturas lógicas, que fossem capazes de representar expressões matemáticas, de uma forma eficiente e totalmente inequívoca, e que permitissem resolver simbolicamente essas expressões de uma forma lógica e computacional. A estrutura triangular e transformação lógica por reescrita axiomática foram o resultado desse estudo.

Pudemos concluir, desse estudo, que era possível resolver, de forma simbólica, um certo tipo de expressões matemáticas, representadas de forma binária, aplicando, de forma sistemática e lógica, um certo número de transformações lógicas sobre estruturas triangulares bem definidas.

Esse tipo de estrutura permitiu que fosse possível classificar essas

expressões com base na estrutura triangular da sua componente principal e, simplificá-las, componente a componente, através da transformação e composição lógica dessas componentes. A transformação lógica é feita por reescrita axiomática, ou seja, através da aplicação de regras de reescrita com estruturas triangulares bem definidas.

As regras de reescrita estão organizadas por poder redutor e definem a reescrita axiomática como redutora ou potencialmente redutora. A forma normal de uma expressão é simplesmente uma forma irredutível dessa expressão. A resolução simbólica de expressões é terminável e conduz sempre à forma mais simples se a base axiomática for completa.

Em suma, podemos dizer que

- As reduções são transformações lógicas que resultam da aplicação directa de fórmulas matemáticas e que conduzem sempre à forma mais simples dessas expressões. As reduções são as transformações lógicas com o poder redutor mais elevado;
- As conversões são transformações lógicas que resultam da aplicação directa de fórmulas matemáticas mas que conduzem apenas a formas, redutoras ou potencialmente redutoras, dessas expressões. O seu poder redutor depende do tipo de regra que for aplicada;
- As simplificações são transformações lógicas que resultam da aplicação indirecta de reduções e conversões ou da aplicação recursiva de outras simplificações;

As expressões matemáticas, que foram objecto deste estudo, estão representadas por expressões lógicas, ou seja, por expressões com estruturas triangulares bem definidas cuja leitura operacional é totalmente inequívoca. As estruturas triangulares são estruturas arbóreas binárias. A leitura operacional

confere às expressões um significado operacional e portanto, uma leitura da sua estrutura lógica.

As expressões matemáticas são completamente caracterizadas pela sua assinatura, ou seja, pela estrutura triangular da sua componente principal. O poder redutor de uma estrutura depende apenas da regra que lhe for aplicada. A reescrita de uma expressão depende, portanto, da sua assinatura e do poder redutor das fórmulas matemáticas que constituem a sua base axiomática.

# Capítulo 3 O Enquadramento Teórico

## A Representação Lógica e a Reescrita Condicional de Expressões Matemáticas

## 3.1 Introdução

O nosso trabalho de investigação está relacionado com a área de computação algébrica (CA) [15] e com a reescrita de expressões matemáticas por via estritamente axiomática utilizando lógicas de 1ª ordem.

As operações matemáticas foram estudadas ao seu nível axiomático. A esse nível, as expressões matemáticas estão representadas por expressões lógicas. Considerámos as operações matemáticas como o resultado de reescritas axiomáticas, ou seja, o resultado da aplicação de um certo número de axiomas e teoremas. Numa perspectiva axiomática, a resolução simbólica de expressões matemáticas não é mais do que o resultado de uma aplicação sistemática de um certo de reescritas axiomáticas envolvendo as expressões e as suas equivalentes. As estratégias aplicadas são apenas um reflexo do raciocínio empregue na resolução.

O que, principalmente, nos motivou para este trabalho de investigação foi o facto de não existirem muitos sistemas de computação algébrica (CAS) capazes de expor com detalhe os raciocínios empregues, ou seja, produzir soluções detalhadas de cálculos simbólicos, com passos pedagogicamente relevantes. Característica que

[15] Termo que é referido na literatura inglesa por "*Computer Algebra System*".

para Beeson[16] [BEE98] é crucial para o desenvolvimento de sistemas CAS, especialmente, orientados para o ensino da Matemática. O sistema MathXpert™ e o sistema desenvolvido por Ravaglia et al. [RAV98] ("*Derivation System*") para o programa EPGY da Universidade de Stanford são dois sistemas CAS desenvolvidos com essa característica. Foram ambos desenvolvidos com fins estritamente educacionais. O Mathematica™ e o Maple™, por outro lado, dois dos maiores e mais conhecidos sistemas CAS, não possuem essa característica. Esses sistemas foram, essencialmente, desenvolvidos para dar respostas concretas a problemas colocados por especialistas e não para expor o tipo de raciocínio que é normalmente empregue na resolução desses problemas. Esses sistemas são utilizados também no ensino mas, essencialmente, como ferramentas de cálculo.

A nossa abordagem, neste trabalho de investigação, foi também baseada em soluções, mas orientada, essencialmente, para o tratamento axiomático de expressões com estruturas arbóreas binárias, ou seja, a resolução simbólica de expressões por reescrita axiomática.

[16] "*In order to allow the creation of fine-grained solutions, we need some very weak symbolic operations*" (Original).

## 3.2 Computação Algébrica

A computação algébrica (CA) é uma área da Matemática e da Ciência de Computação que se interessa essencialmente pelo desenvolvimento, implementação e aplicação de algoritmos que analisam e manipulam expressões matemáticas. Segundo Cohen [COH03, COH02], os sistemas CAS têm por objectivo principal a resolução simbólica de expressões matemáticas, por meios meramente computacionais. Esse tipo de resolução envolve algoritmos bastante poderosos e sofisticados que operam apenas sobre uma estrutura arbórea ("*tree structure*"), não necessariamente binária, dessas expressões, obtida por simplificação automática. Muitas das transformações efectuadas, por simplificação automática, são de relevância pedagógica, mas inacessíveis fora desses algoritmos. De facto, todos os operadores, matemáticos ou linguísticos (comandos), operam apenas sobre essa forma simplificada de expressões. Segundo Cohen [COH03, COH02], a simplificação automática é baseada na aplicação de um certo número de regras de transformação baseadas nos axiomas da Álgebra e Trigonometria. Essas regras fazem parte de poderosos algoritmos de simplificação e não podem ser acedidas, publicamente, como operações elementares ou atómicas.

Segundo Cohen [COH03, COH02], nos últimos 35 anos, a investigação nesta área tem-se concentrado, essencialmente, no desenvolvimento de algoritmos, eficazes e eficientes, para muitas das operações utilizadas em Matemática. Entre elas estão incluídas, por exemplo, a decomposição e factorização de polinómios. Porém, o que nos motivou para este estudo não foram os polinómios mas sim a necessidade de tornar públicas as operações elementares que estão normalmente envolvidas na simplificação algébrica das expressões matemáticas, mais comuns. Segundo Cohen [COH03, COH02], muitas das operações elementares da Álgebra e do Cálculo podem ser implementadas, representando as expressões matemáticas como estruturas arbóreas simplificadas e utilizando algumas operações primitivas para as analisar e construir através de algoritmos. Só que, segundo Beeson

[BEE98], a grande maioria desses algoritmos não produz respostas, pedagogicamente, relevantes. De facto, muitas das operações simbólicas que consideramos atómicas ou elementares, não são acessíveis através desses algoritmos.

Consideramos que

> **as computações efectuadas ao nível axiomático são operações atómicas ou elementares.**

De facto, as operações simbólicas efectuadas a esse nível são o resultado da aplicação de axiomas e/ou teoremas e, portanto, indivisíveis. No entanto, na maioria dos sistemas de computação algébrica, essas operações não podem ser explicadas ao nível axiomático. Como veremos, a reescrita axiomática é uma operação simbólica que podemos considerar atómica ou elementar.

Segundo Beeson [BEE98], a maioria das experiências realizadas, até à data, na área de instrução programada, tanto em Álgebra como em Cálculo, tem sido feita utilizando, essencialmente, programas de computação algébrica (CAS) como o Mathematica™ e o Maple™. Programas que, na opinião de Beeson, não foram especialmente concebidos para fins educacionais. Beeson considera que não é possível obter resultados, com alguma relevância pedagógica, ou a necessária granularidade, adicionando apenas novas funcionalidades às interfaces desses sistemas. A concepção desses sistemas possui ramificações computacionais tão profundas que é praticamente impossível desassociar as suas interfaces, dos aspectos funcionais dos seus núcleos. Beeson considera que só é possível desenvolver essas componentes separadamente, se os próprios núcleos forem também concebidos com os mesmos objectivos em mente. Pois, só assim se poderá evitar que muitas das capacidades funcionais desses sistemas façam parte apenas de poderosos, mas impenetráveis algoritmos.

Segundo Beeson [BEE98], um dos trabalhos, mais sérios e exaustivos,

realizados nessa área, foi sem dúvida o desenvolvimento do sistema de derivação ("*Derivation System*"), desenvolvido por Ravaglia et al. [RAV98], para o programa EPGY ("*Education Program for Gifted Youth*") da Universidade de Stanford.

O sistema de derivação utiliza o Maple™ como "*motor inferencial*". O sistema é apenas a componente pedagógica, ou "*máquina semântica*", que é interposta entre o Maple™ e o utilizador. Segundo Beeson [BEE98], para o desenvolvimento dessa componente, foi necessário despender um enorme esforço de programação, tanto em C como em Maple™, para se conseguir colmatar muitas das deficiências do Maple™, nessa área. O próprio Ravaglia [RAV98] reconhece que, só por si, o Maple™ não seria capaz de satisfazer todos os objectivos operacionais a que ele e a sua equipa se tinham proposto.

Beeson [BEE98] considera também que só é possível obter soluções, com alguma relevância pedagógica, se os sistemas, em questão, forem capazes de reproduzir operações simbólicas a níveis elementares ou atómicos. Níveis que considerámos, nesta tese, como axiomáticos. Ravaglia [RAV98] reconhece também que para desenvolver sistemas desse tipo é necessário possuir um controlo total sobre todas as capacidades computacionais desses sistemas.

O sistema desenvolvido por Ravaglia produz soluções desse tipo, mas não de uma forma automática. O sistema desenvolvido por Beeson, por outro lado, fá-lo de uma forma semelhante à nossa. Só que no nosso caso, as expressões são identificadas com base na assinatura e não através do tópico onde se enquadram.

Segundo Ravaglia [RAV98], uma das maiores fraquezas dos sistemas CAS, como ferramentas pedagógicas, advém paradoxalmente do seu enorme poder computacional e da sua generalidade de propósitos. Beeson [BEE98] considera ainda que sistemas, como o Maple™ e o Mathematica™, utilizam algoritmos, baseados em métodos de cálculo que são, por vezes problemáticos, do ponto de vista pedagógico. Para Beeson, o que os torna inadequados como ferramentas

pedagógicas é precisamente a sua incapacidade de decompor as computações em passos mais compreensíveis. Beeson [BEE98] vai ainda mais longe quando afirma que, mesmo no caso de soluções que podem ser obtidas por meios elementares, é importante que elas soluções possam ainda ser decompostas em passos mais simples.

Mas nada do que foi dito sobre o Mathematica™ e o Maple™ , diminui a enorme importância desses sistemas, ou o enorme contributo que têm prestado ao estudo da computação simbólica. Buchberger [BUC96] considera o Mathematica™ como um dos mais avançados e completos sistemas CAS que, até à data, têm sido desenvolvidos. Para além de uma linguagem de programação, moderna e bastante avançada, o Mathematica™ possui também uma vasta biblioteca de poderosos algoritmos matemáticos. Segundo Buchberger [BUC96], essa é a principal razão porque a maioria dos seus utilizadores a utiliza. Aspectos que muitos outros investigadores [BEE98, KAJ98, RAV98] consideram como não sendo os mais adequados do ponto de vista pedagógico.

Um outro aspecto importante que deve ser abordado também, é o tipo de interface disponibilizado por esses sistemas aos seus utilizadores. Kajler e Soiffer [KAJ98] consideram que um dos problemas relacionados com esses sistemas é, precisamente, a forma linear e, segundo eles, contra-natura, como as expressões matemáticas são introduzidas nesses sistemas.

Nas palavras de Kajler e Soiffer [KAJ98],

> **"These problems may intimidate novice users and frustrate experienced users."** [17]

Esse é precisamente o tipo de interface que é utilizado pelo MathXpert™. O sistema de derivação, desenvolvido por Ravaglia et al. [RAV98], utiliza uma

[17] "*Estes problemas podem intimidar os utilizadores menos habilitados, ou mesmo frustra os mais experientes.*" (Tradução)

interface gráfica. No nosso caso é utilizada uma interface baseada na linguagem corrente.

Uma interface gráfica torna mais fácil atingir esse objectivo, mas não é a única forma de o fazer. A escrita que designamos por manuscrita ("*handwritten*"), por exemplo, é uma outra forma de o fazer. Esse tipo de escrita utiliza um outro tipo de interface.

Um outro aspecto importante, relacionado com este tipo de sistemas, é o facto da linguagem utilizada em muitos deles ser, essencialmente, uma linguagem de programação. Tal como a Matemática, as linguagens de programação são também linguagens, por natureza, formais e abstractas. Do ponto de vista pedagógico, não se justifica que uma seja substituída pela outra. De facto, nem todos os potenciais utilizadores desses sistemas são, ou pretendem ser, exímios programadores. Segundo Kajler e Soiffer [KAJ98], quanto mais natural e intuitiva for a linguagem desses sistemas, tanto maior será a probabilidade deles virem a ser utilizados em computações e derivações.

A escolha que fizemos assenta precisamente nesse princípio. Tanto a linguagem corrente da Matemática (*natural*) como uma forma abreviada dessa linguagem (*quasi-natural*) são formas de escrita que consideramos naturais e intuitivas. De facto, oralmente, não existe outra forma de comunicação Matemática que não a corrente. Escritas nessa forma, as expressões matemáticas são lidas de uma forma inequívoca.

Segundo Beeson [BEE98], o modelo proposto por Buchberger (1990), para este tipo de sistemas, é o modelo conhecido por "*white box/black box*". Buchberger considera uma "*black box*" como uma solução com apenas um passo, e uma "*white box*", como uma solução com vários passos. De acordo com esse modelo, um algoritmo funciona como uma "*black box*". Beeson considera as soluções com

vários passos como "*glass boxes*", e cada passo como uma "*black box*".

A maioria dos sistemas CAS funcionam de acordo com esse modelo. O tamanho da computação, ou seja, a granularidade da operação, pode variar de sistema para sistema. Note que quanto mais fina for essa granularidade, tanto maior será a capacidade do sistema de operar ao nível axiomático. De facto, Beeson [BEE98] considera que um sistema de computação simbólica só é capaz de produzir soluções simbólicas, de uma certa granularidade, se for capaz de operar simbolicamente a esse nível, ou seja, reproduzir de uma forma simbólica, operações elementares, ou atómicas, que ele apelidou de *fracas*.

Como veremos na Secção 4.4.2, a resolução simbólica de expressões é também baseada nesse modelo. Cada passo da resolução é produzido por reescrita axiomática. Como veremos, a reescrita axiomática é uma transformação lógica que resulta da aplicação de axiomas ou teoremas sob a forma de reescrita. Uma reescrita axiomática como, por exemplo, $x+0 \to x$, é considerada uma "*black box*". Uma solução simbólica, por outro lado, é uma sequência de transformações lógicas, ou reescritas axiomáticas. Uma solução simbólica como, por exemplo, $\log_e 4^2 \to 2\log_e 4 \to 2\log_e 2^2 \to \cdots$, é considerada uma "*glass ou white box*". No caso da resolução simbólica, a reescrita axiomática é a *operação simbólica* mais atómica ou elementar que se pode efectuar sobre uma expressão matemática. Isto porque uma transformação lógica só pode ocorrer ao nível axiomático. Por exemplo, uma transformação lógica, como $2\log_e 4 \to 2\log_e 2^2$, resulta da factorização de um inteiro (Secção 4.3.1.4). Essa transformação pode ser considerada, portanto, uma operação atómica ou elementar.

Segundo Buchberger [BUC96], o próprio Mathematica™ utiliza uma linguagem de reescrita condicional, de ordem superior, como linguagem de expressões. As ideias básicas sobre reescrita remontam a Axel Thue (1914) (cit. [DER01]). Segundo Dershowitz et al. [DER01], a reescrita tem o mesmo poder

computacional que as máquinas de Turing. Dershowitz et al. [DER01] considera a computação simbólica como a reescrita de expressões na sua forma normal. A reescrita de expressões é um tema que é abordado mais à frente na Secção 3.5.

Segundo Beeson [BEE98], muitas das operações simbólicas não são exprimíveis como regras de reescrita. Isto porque o número de argumentos envolvidos é, em geral, arbitrário e alguns desses argumentos poderem estar envolvidos noutras operações. Beeson [BEE98] considera que operações desse tipo devem ser representadas por funções, o que obriga, naturalmente, à utilização de lógicas de ordem superior. Segundo Dershowitz et al. [DER01], para se garantir a convergência desses sistemas, são necessárias técnicas de unificação mais avançadas como, por exemplo, a unificação semântica. Esse tema é tratado em [BAA01]. Tanto Beeson [BEE98] como Dershowitz et al. [DER01] afirmam que a comutatividade e a associatividade podem introduzir dificuldades acrescidas ao processo de reescrita.

Nesta tese, as operações simbólicas estão representadas por termos de 1ª ordem. Como vimos na Secção 2.2, as expressões matemáticas estão também representadas por termos de 1ª ordem. Ambas exibem estruturas triangulares bem definidas, o que tornou possível a implementação de um sistema de reescrita axiomática com base nesse tipo de estrutura. Como vimos na Secção 2.4, uma estrutura triangular não é mais do que uma estrutura arbórea binária, com uma assinatura bem definida. Como veremos na Secção 4.3, expressões com esse tipo de estrutura podem ser reduzidas, simplificadas ou convertidas por reescrita axiomática. Esse processo é baseado no método de resolução SLDNF[18] do Prolog e designado, nesta tese, por resolução simbólica de expressões, por reescrita axiomática.

[18] A sigla SLDNF ("***S**elected, **L**inear Resolved, **D**efinite Clauses, **N**egation by **F**ailure*") refere-se às iniciais do nome que é dado, em inglês, a este método, ou seja, "***L**inear **R**esolution for **D**efinite Clauses with **S**election Function and **N**egation by **F**ailure*".

As expressões matemáticas estão representadas, nesta tese, por termos Prolog, e as regras de reescrita, por cláusulas Prolog. A resolução simbólica de expressões produz soluções que reflectem, de uma forma completa e detalhada, o raciocínio lógico (ou dedutivo) empregue na simplificação dessas expressões, por reescrita axiomática.

Segundo Wos et al. (1984) (cit. [LUG93]), um programa capaz de *raciocinar*[19] é um programa que

> **"employs an unambiguous and exacting notation for representing information, precise inference rules for drawing conclusions, and carefully delineated strategies to control those inference rules"** [20]

O Assistente de Matemática que desenvolvemos é um programa capaz de raciocinar com base em estratégias e regras de reescrita, no âmbito do ensino e aprendizagem da Matemática, ao nível do secundário. O Assistente não é um "*tutor electrónico*", no sentido em que esse termo é definido na literatura. Os "*tutores electrónicos*" são sistemas de tutoria electrónica com uma forte componente pedagógica. Uma das suas características é precisamente a sua adaptabilidade às necessidades de aprendizagem de cada aluno. Os sistemas de tutoria electrónica são conhecidos, na literatura da especialidade, por "*Sistemas de Tutoria Inteligentes*"[21] .

Segundo Nicaud (1994) (cit. [RAV98]), os "*tutores electrónicos*" são sistemas difíceis, ou mesmo impossíveis, de construir. A dificuldade está precisamente na adaptação do sistema ao aluno ("*student model*"), ou seja, na criação de uma componente pedagógica que seja adaptável às necessidades de

[19] Termo referido na literatura inglesa por "*automated reasoning*".

[20] "*emprega uma notação exacta e inequívoca para representar informação, regras de inferência precisas para tirar conclusões e estratégias cuidadosamente delineadas para controlar essas regras*" (Tradução).

[21] Na literatura inglesa, estes sistemas são normalmente designados por "*Intelligent Tutoring Systems*", ou sistemas "*ITS*".

aprendizagem de cada aluno. O interesse da comunidade científica por este tipo de sistemas tem, portanto, diminuído significativamente nos últimos anos. Os "*tutores cognitivos*" de Anderson [AND95] são um exemplo deste tipo de tutores.

A componente "*inteligente*" desses tutores, ou seja, a sua componente especialista, pode ser desenvolvida separadamente, e o sistema disponibilizado como um "*assistente electrónico*". Os "*assistentes electrónicos*" são sistemas especializados em determinadas áreas do conhecimento e portanto com capacidade de raciocínio nessas áreas. A capacidade de raciocínio de um tutor é aquilo que normalmente designamos por "*motor inferencial*". Os "*assistentes electrónicos*" não possuem qualquer capacidade pedagógica. Proporcionam apenas os meios electrónicos através dos quais os alunos podem testar as suas competências nessas áreas.

Um dos Assistentes de Matemática comercializados com algum sucesso nesta área foi sem dúvida o sistema desenvolvido por Beeson[22]. O sistema foi apresentado à imprensa em 2000 sob o nome de MathXpert™ [23]. O sistema desenvolvido por Ravaglia é apenas acessível aos alunos inscritos no programa. Não é possível obter esse sistema através dos canais normais de distribuição e de revenda ao público. O sistema é apenas uma ferramenta de aprendizagem disponibilizada aos alunos inscritos no programa EPGY da Universidade de Stanford.

O Assistente que desenvolvemos é funcionalmente semelhante ao MathXpert™, mas desenvolvido totalmente em Prolog. Tal como Beeson, também nós constatámos que, entre a lógica e a computação simbólica, existe de facto uma forte ligação. As expressões matemáticas, incluídas nesta tese, podem ser

[22] Beeson começou a desenvolver o Mathpert, nome que foi mais tarde alterado para MathXpert, em 1985 tendo levado 11 anos a completá-lo. Só em 2000, porém, é que o sistema foi comercializado.

[23] O precursor era conhecido por Mathpert.

resolvidas simbolicamente de uma forma puramente lógica. A nossa abordagem é, porém, uma abordagem baseada numa lógica de 1ª ordem. Para essa abordagem contribuiu, de forma significativa, o conceito de estrutura triangular e de reescrita axiomática que desenvolvemos para esta tese. A eficácia e a enorme flexibilidade do nosso Assistente de Matemática, assim o atestam.

Tanto Beeson [BEE98] como Ravaglia [RAV98] consideram que um dos problemas com sistemas, como o Mathematica™ e o Maple™, é a aparente facilidade com que nesses sistemas se podem cometer erros lógicos elementares e inferir, como consequência, resultados contraditórios, ou mesmo absurdos. Um dos exemplos clássicos que é citado por ambos é o exemplo relacionado com a resolução da equação $x = 0$. Nesses sistemas é aparentemente fácil chegar-se à contradição $1 = 0$. Esse tipo de erro é, nitidamente, uma consequência da falta de rigor matemático que muitos desses sistemas exibem ao seu nível mais elementar. Nesta tese não são tratadas equações e, portanto, este problema não foi abordado. No entanto, ele serve como exemplo do que pode acontecer quando operações simbólicas são executadas não tendo em conta a contextualização de certas operações.

Como o principal objectivo desses sistemas não é pedagógico, erros desse tipo não ocorrem frequentemente. As pessoas a quem esses sistemas se destinam já são suficientemente especializadas para não cometerem esse tipo de erro, ou no caso de os cometerem, de os reconhecer como tal e, portanto, ignorá-los. Esses sistemas utilizam algoritmos bastante sofisticados que não foram concebidos, obviamente, para lidar com esse tipo de problemas. A sua utilização por pessoas não especializadas, como é o caso de alunos e outros utilizadores com pouca ou nenhuma experiência em Matemática, pode ser prejudicial, especialmente se o objectivo dessa utilização for estritamente pedagógica. Segundo Beeson [BEE03], trata-se de um erro lógico que versões mais recentes do Mathematica™ têm tentado corrigir, restringindo a aplicabilidade de certas transformações, ou permitindo que o

utilizador especifique certas suposições, como argumentos dessas transformações. Mas, segundo Beeson [BEE03], isso não resolve o problema. Para o resolver, é necessário combinar lógica e computação no mesmo programa. O Mathematica™ não possui métodos de verificação sistemática das condições que devem ser impostas a essas transformações. No MathXpert™, o problema foi resolvido, impondo condições iniciais à execução de certos comandos.

No Assistente,

> **as regras de reescrita são condicionais e, portanto, a sua aplicabilidade dependente da resolução dessas condições.**

Nesta fase da investigação, não tivemos que lidar com esse problema, em particular, pois lidamos apenas com expressões e não com equações. Mas mesmo que o tivéssemos, o problema não teria ocorrido, pois não seria possível transformar uma variável, como $x$ , por reescrita axiomática.

O Assistente difere do MathXpert, na forma como as expressões são introduzidas e editadas. No MathXpert™, as equações só são introduzidas depois de escolhido o tópico. O MathXpert™ oferece, à partida, uma equação típica desse tópico que pode ser reeditada dentro de certos "*limites*"[24]. O MathXpert™ impõe, também, restrições quanto ao tipo e tamanho que essas equações podem assumir.

No Assistente,

> **as expressões podem ser introduzidas de uma forma totalmente livre e sem quaisquer tipos de restrições.**

A sua resolução depende apenas da existência ou não de uma base axiomática para esse tipo de expressões.

No MathXpert™, uma expressão como $\log_e \sqrt{4^3}$ é escrita como

[24] Os limites impostos são do foro interno do MathXpert.

ln(sqrt(4^3)). Essa é a forma que Kajler e Soiffer [KAJ98] consideram como "*contra-naturas*".

No Assistente,

> **as expressões são escritas em linguagem corrente, de uma forma natural ou quasi-natural (abreviada).**

Por exemplo, a expressão $\log_e \sqrt{4^3}$ é escrita como o "*logaritmo natural da raiz quadrada do cubo de 4*" ou, de uma forma quasi-natural, mais abreviada, como "*log e raiz 2 potência 3 4*".

Para muitos, essa forma de escrita, natural ou quasi-natural, pode não ser a ideal, pois é demasiado longa e verbosa e, portanto, susceptível de causar erros de escrita, que nada têm a ver com a escrita, propriamente dita, dessas expressões. Não há dúvida que isso é verdade para quem já possui conhecimentos suficientes sobre a Matemática. Para essas pessoas, a forma gráfica, mais "compacta", seria, de certeza, a mais ideal, pois essa é, precisamente, a forma "*oficial*" de se transmitir esse tipo de conhecimentos nos meios académicos, científicos ou profissionais. Mas, mesmo para essas pessoas, a notação matemática é, apenas, a forma "*natural*" de se comunicar Matemática, por "*escrito*". A forma "*natural*" de se "*falar*" Matemática continua a ser a linguagem natural ou corrente. Não faria sentido "*falar*" Matemática de outra forma. A verbosidade da linguagem corrente afecta apenas a transmissão "*escrita*" de uma expressão. A transmissão "*oral*" pode ser feita, sem dificuldade, em linguagem corrente. E assim, consideramos a linguagem corrente como uma forma adequada de se "*falar*" Matemática com um "*assistente electrónico*".

A escrita gráfica é feita utilizando editores gráficos, como o MathType™ ou o WebEq™. Para as utilizar, é necessário possuir apenas um conhecimento adequado de como essas ferramentas funcionam. Qualquer um seria capaz de

"*reproduzir*" uma expressão matemática, utilizando uma dessas ferramentas, se tivesse ao seu dispor uma "*imagem*" dessa expressão. Porém, no nosso entender, isso não seria "*escrever*" Matemática. Escrever Matemática é saber o que se está a escrever. "*Reproduzir*" uma expressão matemática não é o mesmo que saber escrevê-la.

Por exemplo, será que um aluno é capaz de "*escrever*" uma expressão, como $\log_e \sqrt{4^3}$ , a partir de uma expressão, como "*o logaritmo natural da raiz quadrada do cubo de 4*", escrita em linguagem corrente? Será que o aluno é capaz de "*ler*" essa expressão, a partir da sua escrita em notação matemática? Consideramos a "*escrita*" em linguagem corrente como uma forma apropriada de se "*ler*" uma expressão matemática. Um aluno que saiba "*ler*" em notação matemática e "*escrever*" em linguagem corrente deve saber, com certeza, o que está a "*ler*" ou a "*escrever*".

Optámos, nesta tese, pela linguagem corrente (natural ou quasi-natural), como forma de "*escrita*". O aluno "*escreve*" em linguagem corrente, o que "*lê*" em notação matemática, e o computador faz, exactamente, o oposto. Ou seja, "*lê*" em linguagem corrente e "*escreve*" em notação matemática.

No MathXpert™, os resultados estão representados numa linguagem, reconhecível apenas pelo MathXpert™. A distribuição desses conteúdos só pode ser feita entre utilizadores do MathXpert.

No Assistente,

> **os resultados são representados em *XML/MathML*, numa linguagem, universalmente aceite e reconhecível por qualquer "*browser*" que seja compatível com essas normas.**

A distribuição desses conteúdos pode ser feita entre utilizadores com

"*browsers*" desse tipo.

## 3.3 A Representação Lógica de 1ª Ordem

A Matemática é uma das áreas onde tanto o conhecimento (axiomas e teoremas) como a própria resolução de problemas, neste caso, a simplificação de expressões, podem ser tratadas através da lógica e computação. Em particular, a área do ensino e aprendizagem da Matemática, oferece oportunidades únicas para a aplicação do raciocínio dedutivo.

Para a representação lógica de expressões matemáticas, escolhemos o Prolog como suporte linguístico. Sterling e Shapiro [STE00] consideram o Prolog uma linguagem de programação lógica de 1ª ordem, bastante eficiente, computacionalmente.

Para Pereira e Shieber [PER87], resolver problemas, de uma forma lógica, é procurar soluções que demonstrem, de uma forma construtiva, que esses problemas são consequências lógicas de um certo número de premissas, formalizadas nessa mesma lógica. Esse conjunto de premissas pode ser encarado como um programa, e a resolução como uma computação desse programa. Programação lógica[25] é, basicamente, isso. Mas para isso, é necessário que o processo de busca seja, exaustivo e objectivo, isto é, que termine apenas quando não existirem, de facto, mais soluções, e que inclua apenas as derivações que contribuíram para esse objectivo. Pereira e Shieber [PER87] consideram a integralidade da busca e a orientação por objectivos, como objectivos bastante difíceis de se alcançar, mas que se tornam mais viáveis em linguagens lógicas mais *fracas*. No caso das linguagens de programação lógica, o conceito de cláusula definitiva ("*definite clause*") oferece

[25] O termo "*programação lógica*" ("*logic programming*"), é devido a Robert Kowalski (1974). O termo é utilizado para designar o uso da lógica como linguagem de programação.

um bom compromisso entre expressividade e eficiência computacional. Esse conceito está parcialmente implementado na linguagem Prolog.

Como linguagem de representação lógica, o Prolog é uma linguagem bastante eficaz. Tanto as expressões como as fórmulas matemáticas estão representadas por expressões lógicas (Secção 2.2.1).

Numa linguagem lógica de 1ª ordem (FOL[26]), todas as proposições válidas são representadas por fórmulas atómicas. Uma fórmula atómica é uma expressão do tipo $p(t_1,...,t_n)$, onde $p$ representa um símbolo de predicado, de aridade $n$, e $t_1,\ldots,t_n$, $n$ termos.

**Definição 3.3-1** ***Símbolos de Função e Constantes***

> *Seja $\Omega$ um conjunto de símbolos de função, onde cada $f \in \Omega$, está associado a um numero natural $n \in \mathbb{N}$, designado por aridade de $f$. O conjunto de todos elementos de aridade $n$ é definido por*
>
> $$\Omega^n = \{ f \in \Omega \mid n \in \mathbb{N}, aridade(f, n) \}$$
>
> *Elementos de aridade $0$, $c \in \Omega^0$, são designados símbolos constantes.*

Termos são formados a partir de constantes, variáveis e símbolos de função. Constantes são termos que não podem ser substituídos. Nesta tese, esses termos são designados por atómicos (Secção 2.2.1). Variáveis são símbolos que podem intervir em substituições, ou seja, serem substituídos por termos. Símbolos de função são símbolos que definem termos mais complexos. Nesta tese, esses termos são designados por funcionais (Secção 2.2.1).

[26] Sigla que, na literatura inglesa, significa "***F**irst **O**rder **L**ogic*".

**Definição 3.3-2** ***Termos***

> *Seja* $\mathcal{V}$ *um conjunto de elementos tal que* $\Omega \bigcap \mathcal{V} = \varnothing$. *Cada elemento* $x \in \mathcal{V}$ *é designado variável. O conjunto de todos os termos* $\mathcal{T}(\Omega, \mathcal{V})$ *construídos a partir de* $\Omega$ *e* $\mathcal{V}$, *é definido indutivamente como*
>
> *i)* $\mathcal{V} \subseteq \mathcal{T}(\Omega, \mathcal{V})$, *e*
>
> *ii)* $\forall n \in \mathbb{N}$, $f \in \Omega^n$ *e* $t_1, \ldots, t_n \in \mathcal{T}(\Omega, \mathcal{V})$ , *tem-se que* $f(t_1, \ldots, t_n) \in \mathcal{T}(\Omega, \mathcal{V})$

Um termo $t \in \mathcal{T}(\Omega, \mathcal{V})$ é considerado livre se contiver uma ou mais variáveis, isto é, $\mathcal{V}ar(t) \neq \varnothing$, onde $\mathcal{V}ar(t)$ é o conjunto de variáveis em $t$. O termo $t$ é considerado fechado se $\mathcal{V}ar(t) = \varnothing$. O conjunto de todos os termos constantes é denotado por $\mathcal{T}(\Omega, \varnothing)$ ou simplesmente $\mathcal{T}(\Omega)$. Nesta tese, esse conjunto é denotado por $\mathcal{T}$.

Uma fórmula $p(t_1, ..., t_n)$ é considerada fechada se todos os seus termos forem fechados, isto é, $\mathcal{V}ar(t_i) = \varnothing$, com $i = 1, \ldots, n$. Uma frase atómica é uma fórmula fechada. Uma fórmula atómica é uma fórmula que contém pelo menos uma variável.

Fórmulas são formadas a partir de constantes, variáveis, símbolos de função e símbolos de predicado. Uma fórmula atómica não pode, portanto, conter outras fórmulas.

**Definição 3.3-3** ***Símbolos de Predicado***

> *Um símbolo de predicado* $p$, *de aridade* $n$, *denotado por* $p/n$, *é um símbolo que define uma relação entre* $n$ *termos.*

Segundo Pereira e Shieber [PER87], muitos dos métodos de dedução automatizada operam apenas sobre fórmulas atómicas representadas na forma designada por clausal. Uma cláusula de 1ª ordem é uma disjunção de literais representando fórmulas atómicas ou a sua negação. Todas as variáveis presentes numa cláusula de 1ª ordem estão universalmente quantificadas.

**Definição 3.3-4** ***Cláusula de 1ª Ordem***

> *Uma cláusula de 1ª ordem é uma disjunção de literais, ou seja, uma expressão escrita na forma*
>
> $$P_0 \vee P_1 \vee \cdots \vee P_n \vee \neg N_0 \vee \neg N_1 \vee \cdots \vee \neg N_m$$
>
> *onde $P_i$, representa um literal positivo, e $\neg N_i$, um literal negativo.*

Pereira e Shieber [PER87] consideram que a utilidade desta forma de representação está, precisamente, no facto de qualquer fórmula fechada $\phi$ poder ser transformada mecanicamente numa conjunção de cláusulas $\mathcal{P}$, tal que $\phi$ só é inconsistente[27] se $\mathcal{P}$ também o for. Em geral, essas duas formas de representação, $\mathcal{P}$ e $\phi$, não são equivalentes, pois pode ser necessário introduzir funções *Skolem*[28], para remover os quantificadores existenciais. Mas o importante, segundo Pereira e Shieber, é que a inconsistência é preservada.

As cláusulas $\mathcal{P}$ são normalmente expressas na forma de implicação, ou seja, numa expressão do tipo

$$(N_0 \wedge N_1 \wedge \cdots \wedge N_m) \rightarrow (P_0 \vee P_1 \vee \cdots \vee P_n)$$

[27] Isto é, que contém contradições.

[28] A *skolemização* de uma variável existencial numa fórmula lógica é a sua substituição por uma função que devolve como resultado a constante apropriada mas como função de uma ou mais variáveis nessa expressão. Por exemplo, a variável existencial $Y$, na expressão $\forall X \exists Y$ mother($Y$, $X$), pode ser substituída por uma função *skolem* f($X$) que depende de $X$ e ser escrita como $\forall X$ mother(f($X$), $X$) onde todas as variáveis estão universalmente quantificadas.

Nessa forma, o que a cláusula afirma é que, pelo menos, uma das fórmulas, $P_i$, no seu consequente, só é verdadeira, se todas as fórmulas $N_i$, no seu antecedente o forem também. Uma cláusula sem antecedente, $\rightarrow (P_0 \vee P_1 \vee \cdots \vee P_n)$, é interpretada como uma expressão que é sempre verdadeira, ou seja, como uma verdade e uma cláusula sem consequente, $(N_0 \wedge N_1 \wedge \cdots \wedge N_m) \rightarrow$, como uma expressão que é sempre falsa, ou seja, falsidade. Uma cláusula sem ambos, $\rightarrow$, é interpretada também como falsidade.

O Prolog utiliza uma forma restrita e menos expressiva da forma clausal, conhecida por cláusula Horn. Numa cláusula Horn, só pode existir, quanto muito, um literal positivo [PER87, STE00].

Existem apenas três tipos de cláusulas Horn:

- **Unitárias** – Cláusulas do tipo $\rightarrow P_0$, onde existe apenas um único literal positivo. O Prolog utiliza cláusulas deste tipo para representar verdades absolutas, ou factos;
- **Não unitárias** – Cláusulas do tipo $(N_0 \wedge N_1 \wedge \cdots \wedge N_m) \rightarrow P_0$, onde existe apenas um literal positivo mas um, ou mais, literais negativos. O Prolog utiliza cláusulas deste tipo para representar regras;
- **Negativas** – Cláusulas do tipo $(N_0 \wedge N_1 \wedge \cdots \wedge N_m) \rightarrow$, onde não existe nenhum literal positivo, mas um, ou mais, literais negativos. O Prolog utiliza cláusulas deste tipo para representar perguntas, ou consultas.

As cláusulas positivas (unitárias e não unitárias) são conhecidas por cláusulas definitivas. Essas cláusulas são interpretadas como tendo uma única conclusão definitiva. A conclusão é dada pelo literal positivo do seu consequente.

Cláusulas negativas são conhecidas por cláusulas objectivas. Essas cláusulas

são interpretadas como a negação do seu antecedente. A negação do antecedente, $\neg(N_0 \wedge N_1 \wedge \cdots \wedge N_m)$, pode ser encarada como uma pergunta sobre a verdade desse antecedente. A resposta é obtida por refutação do antecedente, ou seja, demonstrando, por contradição, que a negação do antecedente e o programa $\mathcal{P}$ são inconsistentes, ou seja, que a conjunção $\neg(N_0 \wedge N_1 \wedge \cdots \wedge N_m) \wedge \mathcal{P}$ é falsa. O contra-exemplo fornece a substituição que torna essa conjunção falsa.

A resolvente (Secção 3.3.2) de duas cláusulas definitivas é ainda uma cláusula definitiva. Se, porém, uma delas for negativa (objectiva), então a resolvente (Secção 3.3.2) é também negativa (objectiva). Este facto é de especial relevância para a demonstração de teoremas, por objectivos. Esse é precisamente o paradigma seguido pelo Prolog.

A resolução de um objectivo, em Prolog, envolve a atribuição de valores a variáveis lógicas que tornem esse objectivo, por unificação (Secção 3.3.2) com outras cláusulas do programa, uma consequência lógica desse programa. Podemos dizer que a resolução simbólica de expressões matemáticas não é mais do que a resolução de um certo número de *objectivos*. Esses objectivos estão representados por frases matemáticas. Por exemplo, a resolução simbólica de uma expressão, como $\log_e 4^2$, não é mais do que a resolução de um certo número de objectivos Prolog do tipo query($List$), isto é, objectivos do tipo query([definir,log,e,pwr,4,2]) e query([resolver,prática]). O número de objectivos a resolver depende, naturalmente, da estratégia escolhida (Capítulo 4). O Capítulo 4 mostra, através de uma aplicação prática, como esses objectivos são construídos a partir de frases matemáticas.

Um objectivo Prolog é uma conjunção de cláusulas objectivas. Um programa Prolog é uma disjunção cláusulas definitivas. Os programas Prolog são interpretados de acordo com o menor modelo de Herbrand [PER87, STE00]. Os modelos são interpretações que respeitam a leitura declarativa das cláusulas de um

programa. Uma interpretação do programa é simplesmente um subconjunto da sua base de Herbrand, ou seja, um subconjunto de todos os objectivos fechados que possam ser formados a partir dos seus predicados e do seu universo Herbrand. Um objectivo fechado é um objectivo onde não estão presentes variáveis lógicas. O universo Herbrand de um programa é o conjunto de todos os termos que se podem formar a partir das suas constantes e dos seus símbolos de função. O significado de um programa é o seu modelo mínimo, isto é, a intersecção de todas as suas possíveis interpretações [STE00]. O fecho transitivo de uma relação binária é definível nesse modelo [PER87]. Por exemplo, relações como $\mathrm{r}(A,C) \rightarrow \mathrm{s}(A,C)$ e $\mathrm{r}(A,B), \mathrm{s}(B,C) \rightarrow \mathrm{s}(A,C)$, estão ambas definidas nesse modelo. Note que a base Herbrand pode ser infinita se o seu universo também o for.

O Prolog opera no pressuposto de que o menor modelo Herbrand representa um mundo fechado. Ou seja, que o que não pode ser demonstrado como verdadeiro, é considerado falso. A verdade de uma negação é, por outro lado, interpretada como o insucesso de se demonstrar como verdadeiro aquilo que é negado. A negação por insucesso é uma interpretação da negação lógica ($not\ p$ ou $\sim p$). Esses dois conceitos estão intimamente relacionados. Para o leitor mais interessado, informação sobre Prolog e programação lógica pode ser obtida de várias fontes, como por exemplo [LUG93, STE00].

Em Prolog, o símbolo de implicação é o símbolo "$:-$", com o antecedente no lado direito e o consequente no lado esquerdo. Esse símbolo tem o mesmo significado que o símbolo "$\leftarrow$". Nesta tese, ambos são utilizados com esse significado. Por exemplo, a implicação $\mathrm{r}(a,b) \rightarrow \mathrm{s}(a,b)$, pode ser escrita como $\mathrm{s}(a,b) \leftarrow \mathrm{r}(a,b)$ ou $\mathrm{s}(a,b) :- \mathrm{r}(a,b)$.

### *3.3.1 O Método de Resolução*

O princípio de resolução foi introduzido por Robinson em 1965. Segundo

Bundy [BUN99], o método de resolução pode ser encarado como um método dedutivo, onde novas fórmulas são criadas a partir de outras, por unificação (Secção 3.3.2). O método de resolução (Secção 3.3.1) é uma técnica de demonstração de teoremas utilizada no cálculo de predicados. A demonstração é feita por refutação, ou seja, mostrando que a negação do objectivo contradiz (ou é inconsistente com) as premissas do problema. O Prolog utiliza uma versão restrita desse método, conhecido por método de resolução SLDNF[29]. Este método utiliza uma estratégia de resolução designada por "*linear resolution*" ou "*linear input form strategy*" [LUG93], onde os literais, nas cláusulas definitivas, são seleccionados da esquerda para a direita ("*selection function*") e a negação do literal determinado pelo insucesso da busca ("*negation by failure*"). O método SLDNF opera no pressuposto que o universo é fechado e que tudo o que não é demonstrável, deve ser considerado falso.

Os literais a resolver são seleccionados pelo Prolog, da esquerda para a direita. A função seleccionadora é uma regra computacional $\mathcal{S}$ que opera sobre os literais, ou sub-objectivos $\mathcal{G}$, de cláusulas definitivas $\mathcal{C}$ que são seleccionadas para os resolver. As cláusulas são seleccionadas, de cima para baixo, na mesma ordem sequencial em que estão dispostas no programa.

Suponhamos que o objectivo a resolver, $\mathcal{G}_0$, é uma conjunção de literais, ou sub-objectivos $L_1,\ldots,L_k$. O método de resolução SLDNF constrói para esse objectivo uma cadeia de derivação a partir da resolução de cada um desses literais, ou sub-objectivos. Uma cadeia de derivação é simplesmente uma sequência de objectivos que se pretende resolver. Com a excepção do objectivo inicial, todos os objectivos, nessa cadeia, são objectivos derivados, ou seja, objectivos que são obtidos como resultado da unificação (Secção 3.3.2) de um objectivo com uma

[29] A sigla SLDNF ("***S**elected, **L**inear Resolved, **D**efinite Clauses, **N**egation by **F**ailure*") refere-se às iniciais do nome que é dado, em inglês, a este método, ou seja, "***L**inear **R**esolution for **D**efinite Clauses with **S**election Function and **N**egation by **F**ailure*".

cláusula definitiva do programa. Uma cadeia de derivação pode ser finita, ou infinita. Se designarmos os objectivos por $\mathcal{G}_i$, com $i = 0,\ldots,n$, a cadeia de derivação SLDNF, para o objectivo inicial $\mathcal{G}_0$, é uma sequência do tipo

$$\mathcal{G}_0 \overset{C_0}{\rightarrow}_{\mathcal{S}} \mathcal{G}_1 \overset{C_1}{\rightarrow}_{\mathcal{S}} \cdots \overset{C_{n-1}}{\rightarrow}_{\mathcal{S}} \mathcal{G}_n$$

onde $C_i$, é a cláusula $i$ do programa $\mathcal{P}$, que unifica (Secção 3.3.2) com um literal $L_k$ do objectivo $\mathcal{G}_i$, seleccionado por $\mathcal{S}$, e $\mathcal{G}_n$, o objectivo vazio, que representa uma contradição ou falsidade. O objectivo vazio é representado pela cláusula vazia $[]$.

Para se evitarem possíveis conflitos entre as variáveis, as variáveis em $\mathcal{G}_i$ são automaticamente renomeadas pelo Prolog. Pereira e Shieber [PER87] consideram os mecanismos utilizados pelo Prolog, bastante eficientes. O objectivo $\mathcal{G}_i$ é a resolvente (Secção 3.3.2) do objectivo $\mathcal{G}_{i-1}$ do qual foi derivado. Por exemplo, se $\mathcal{G}_{i-1}$ for a conjunção de literais $L_1,\ldots,L_k$ e $\mathcal{C}_{i-1}$ a cláusula $A :- B_1,\ldots,B_n$, o objectivo derivado $\mathcal{G}_i$ é a resolvente $(L_1,\ldots,B_1,\ldots,B_n,\ldots,L_k)\sigma_{i-1}$ que resulta da unificação de $L_i$ com $A$ (a cabeça ou consequente de $\mathcal{C}_{i-1}$), ou seja, $\mathcal{G}_i = \mathcal{G}_{i-1}\sigma_{i-1}$. A cláusula $\mathcal{C}_{i-1}$ é a cláusula que foi seleccionada pela regra de selecção $\mathcal{S}$ A substituição, $\sigma_{kr} = \text{mgu}(\mathcal{C}_k, L_r)$, é o unificador mais geral (Secção 3.3.2) [STE00], para o literal $L_r$ e o objectivo $\mathcal{G}_k$.

O objectivo inicial $\mathcal{G}_0$ é refutado (ou seja, a sua negação é demonstrada por contradição) se o objectivo final $\mathcal{G}_n$, da sua cadeia de derivação, for o objectivo vazio. Diz-se, então, que $\mathcal{P} \models [\mathcal{G}_0]\sigma_1 \ldots \sigma_n$, ou seja, que o contra-exemplo $[\mathcal{G}_0]\sigma_1 \ldots \sigma_n$ é uma consequência lógica do programa $\mathcal{P}$. O contra-exemplo é simplesmente uma instância do objectivo inicial $\mathcal{G}_0$, que resulta da aplicação da

substituição composta, $\sigma_1 \ldots \sigma_n$.

Uma cadeia de derivação SLDNF pode ser encarada como uma demonstração, por refutação, de que o objectivo é uma consequência lógica do programa, ou seja, que a sua negação conduz a uma contradição com o programa.

Assim, consideramos que

> **o método SLDNF do Prolog preenche todos os requisitos computacionais que uma linguagem de computação simbólica necessita para resolver simbolicamente expressões matemáticas com estruturas triangulares bem definidas.**

De facto, a resolução simbólica de expressões, como linguagem computacional de reescrita de expressões matemáticas, por via axiomática, necessita apenas de

1. Um método de busca para a selecção de fórmulas matemáticas com base na unificação de expressões matemáticas com o lado esquerdo dessas fórmulas;

2. Um método de resolução para a reescrita de expressões matemáticas com base na unificação de expressões matemáticas com o lado esquerdo dessas fórmulas;

3. Uma forma de representação lógica que seja compatível com esses dois métodos e que sirva para representar expressões e fórmulas matemáticas.

O Prolog fornece uma estratégia de resolução linear, por objectivos, com busca de soluções em profundidade e *backtracking* que preenche os dois primeiros requisitos. As expressões matemáticas e o lado esquerdo de fórmulas matemáticas podem ser representados, em Prolog, por termos de 1ª ordem. As fórmulas

matemáticas podem ser representadas por regras Prolog. Essa forma de representação preenche o último requisito, pois são compatíveis com a forma como o Prolog selecciona e resolve objectivos. Os literais a resolver (objectivos) são seleccionados da esquerda para a direita e as cláusulas para os resolver, seleccionadas, de cima para baixo, na mesma ordem em que estão dispostas no programa [STE00]. Por exemplo, a resolução simbólica de uma expressão matemática, pode ter como objectivo original uma conjunção do tipo

$$\text{query}(List_1),\ldots,\text{query}(List_i),\ldots,\text{query}(List_n)$$

onde cada literal representa o processamento de uma frase matemática. A resolução simbólica de expressões não é mais do que uma demonstração, por refutação, de que uma dada sequência de frases conduz ao resultado pretendido. O contra-exemplo fornece uma possível solução simbólica.

O método utilizado pelo Prolog não é determinista. Pode existir mais de que uma resolvente (Secção 3.3.2) para os literais seleccionados. O resultado de uma resolução depende, portanto, da ordem em que os literais e as cláusulas são seleccionadas.

O significado operacional de um programa depende, portanto, da ordem em que os literais e as cláusulas estão dispostas no programa. A ordem dos literais numa cláusula, ou a ordem das cláusulas no programa, podem causar problemas de terminação. Segundo Sterling e Shapiro [STE00], cláusulas com recursividade esquerda, directa ou indirecta, podem originar cadeias de derivação infinitas. O Prolog possui, portanto, algumas fragilidades.

### *3.3.2 O Método de Unificação*

A ideia chave por detrás do método de resolução é, sem dúvida, a unificação. Robinson (1965) foi o primeiro a utilizar o termo "*unificação*" e a investigar formalmente essa noção [BAA01, STE00]. A unificação tem, por

objectivo, encontrar soluções para equações do tipo $s = t$, onde $s$ e $t$ representam termos sintacticamente diferentes. A unificação determina que substituições são necessárias efectuar para tornar esses termos idênticos. As variáveis são substituídas por termos do seu domínio. Por exemplo, se $s = f(a,x)$ e $t = f(y,b)$, onde $x$ e $y$ são duas variáveis lógicas e $a$ e $b$, duas constantes do seu domínio, fazendo $x = b$ e $y = b$, obtém-se o mesmo termo em ambos os casos, ou seja, o termo $f(a,b)$. A substituição tornou-os sintacticamente idênticos. Diz-se então que esses dois termos, $s$ e $t$, são unificáveis e que o termo $f(a,b)$ é o termo que os unifica.

**Definição 3.3-5** ***Termos Unificáveis***

> *Diz-se que dois termos, $s$ e $t$, são unificáveis, se é só se existir uma substituição $\sigma$, tal que $s\sigma \equiv t\sigma$[30]. A substituição $\sigma$ é o unificador desses dois termos.*

Baader e Snyder [BAA01] classificam esse tipo de unificação, de sintáctica ou unificação de 1ª ordem. Nesse tipo de unificação, só podem intervir variáveis lógicas [BAA01, VAL00].

Mas equações do tipo $s = t$ podem ter vários unificadores. Por exemplo, se $s = f(x,y)$ e $t = f(y,x)$, onde $x$ e $y$ representam duas variáveis lógicas, qualquer substituição do tipo $\sigma = \{x \mapsto u, y \mapsto u\}$, com $u$ pertencente ao domínio das variáveis, seria uma substituição unificadora desses dois termos. Mas, segundo Baader e Snyder [BAA01], só são necessários unificadores mais gerais, ou seja, unificadores a partir dos quais todos os outros se podem obter por instanciação.

**Definição 3.3-6** ***Unificador Mais Geral***

> *Diz-se que um unificador $\alpha$ é mais geral que um outro $\beta$, se para uma substituição $\gamma$ e para um par de termos $s$ e $t$, $(s\alpha)\gamma \equiv s\beta$ e*

[30] O símbolo "≡" é o símbolo utilizado, aqui, para representar uma igualdade, estritamente, sintáctica.

> $(t\alpha)\gamma \equiv t\beta$, *ou seja,* $\alpha\gamma \equiv \beta$. *Escreve-se então que* $\alpha = mgu(s,t)$.

Neste caso, diz-se que o unificador $\beta$ é apenas uma instância de $\alpha$. O termo $s\alpha$ é o termo que resulta da aplicação de uma substituição $\alpha$ a uma expressão $s$, ou seja, o termo que resulta da substituição todas as suas variáveis $x_i$, pelos respectivos termos de substituição, $t_i$. Segundo Baader e Snyder [BAA01], o método de unificação não decide apenas se dois termos são ou não unificáveis; computa também o seu unificador mais geral.

**Definição 3.3-7** ***Resolvente***

> *A resolvente* $\sigma$, *escrita na sua forma resolvida, é um conjunto de equações*
>
> $$\sigma = \{x_1 \mapsto t_1, \ldots, x_i \mapsto t_i, \ldots, x_n \mapsto t_n\}$$
>
> *onde* $x_i$ *representa uma variável lógica, e* $t_i$, *o termo que a substitui.*

A resolvente não pode conter termos livres $t_i$, cujas variáveis possam ter intervindo na unificação. Isto é, qualquer que seja $i$ ou $j$, se $x_i \mapsto t_j$, então $x_i \notin \mathcal{V}ar(t_j)$, onde $\mathcal{V}ar(t_j)$ é o conjunto de variáveis em $t_j$. O *occurs_check* é o teste que garante isso, mas que não está presente, por razões de desempenho, na maioria das implementações Prolog. A forma, bastante eficaz, como o Prolog renomeia as suas variáveis, torna esse teste desnecessário na maioria dos casos.

Os métodos de resolução e de unificação tornam possível fazer inferências sobre expressões lógicas com base em fórmulas matemáticas cujos lados esquerdos possuem a mesma estrutura triangular (Secção 2.4.1).

## 3.4 A Representação Lógica de Expressões

Como foi referido atrás na Secção 2.3, a escrita de expressões matemáticas, como estruturas triangulares, em linguagem corrente (natural) ou de uma forma abreviada (quasi-natural), confere um significado operacional, inequívoco, a essas expressões. Segundo Pereira e Shieber [PER87], o desenvolvimento da programação lógica tem estado intimamente ligado à procura de formalismos computacionais de como exprimir a análise sintáctica e semântica de frases escritas numa linguagem natural. Formalismos esses que Pereira e Shieber designam por gramáticas lógicas. As gramáticas lógicas são essencialmente uma forma lógica de formalizar regras gramaticais. Pereira e Shieber consideram que essa formalização pode ser feita através de cláusulas definitivas (cláusulas Horn) e que as gramáticas lógicas podem ser executadas directamente por programas, como o Prolog.

As gramáticas lógicas, baseadas em cláusulas definitivas, são designadas por gramáticas DCG[31]. Este tipo de gramática possui uma forte ligação com a linguagem Prolog. Porém, a implementação Prolog é apenas uma forma restrita do formalismo DCG. O próprio Prolog, como se sabe, é também uma forma restrita do formalismo dedutivo baseado em cláusulas Horn.

Segundo Pereira e Shieber [PER87], uma forma bastante comum de parametrizar a semântica de uma linguagem natural, é associar a cada expressão, escrita nessa linguagem, uma forma de representação lógica cujo significado seja precisamente o mesmo que o atribuído a essa expressão na sua forma natural. Por exemplo, expressões como "*resolver o logaritmo natural do quadrado de 4*", escritas em linguagem corrente, podem ser parametrizadas através de expressões lógicas, como resolve(practice(expression(log(e,pwr(2,4))))) . Pereira e Shieber consideram que só é possível estabelecer esse tipo de associação se a forma lógica associada a

[31] Termo que, na literatura inglesa, representa uma abreviatura de "***D**efinite **C**lause **G**rammar*".

essas expressões puder ser composta também a partir das formas lógicas associadas às suas partes constituintes, ou seja, que a composição ser, de certo modo, semântica.

Como veremos na Secção 4.2.2, as expressões matemáticas, incluídas neste estudo, são formadas por composição lógica de construtores-$\lambda$. Essa forma de composição é bastante semelhante à que é preconizada por Pereira e Shieber.

As expressões são analisadas pela gramática DCG, e convertidas numa forma lógica através do método de composição lógica, preconizado por Pereira e Shieber [PER87], ou seja, uma forma simplificada de composição semântica, inspirada no método de Montague, mas que utiliza apenas lógicas de 1ª ordem. O método utilizado por Montague é baseado no cálculo lambda e utiliza lógicas de ordem superior.

Mas para associar formas lógicas a essas expressões é necessário utilizar o conceito de função. O cálculo lambda seria, por conseguinte, bastante útil, aqui. Mas, como se sabe, esse conceito só é implementável em lógicas de 1ª ordem através de símbolos de função, com a óbvia limitação de que cada símbolo só pode representar uma função.

No cálculo lambda, como se sabe é possível representar funções a partir de outras funções. Por exemplo, a função "*sucessor de* $x$", $f(x) = x + 1$, poderia ser representada por $\lambda x.(x+1)$, em vez de uma expressão como $\text{succ}(x)$, onde é utilizado um símbolo de função. Nessa expressa, o símbolo $\lambda$ serve apenas para identificar o argumento da função. Uma expressão como $(\lambda x.(x+1))(5)$ seria $\beta$-reduzida à sua expressão equivalente, $5+1$, mas não ao valor numérico 6, que ela de facto representa. Segundo Pereira e Shieber [PER87], no cálculo lambda, são apenas efectuadas conversões-$\alpha$ e reduções-$\beta$. Uma redução-$\beta$ é uma forma de transformação que envolve apenas a substituição dos argumentos. É o que, nesta

tese, designamos por substituição da variável. Uma conversão-$\alpha$, por outro lado, é uma forma de transformação que envolve apenas a renomeação desses argumentos. Nesta tese, as conversões-$\alpha$ são efectuadas pelo Prolog.

Para a representação desse tipo de funções, em Prolog, Pereira e Shieber [PER87] sugerem a utilização de um operador de formação de funções, baseado no cálculo lambda. Operador que Pereira e Shieber designam por operador-$\lambda$. Tal como o símbolo $\lambda$, esse operador permite também a representação de funções a partir de outras funções. As expressões onde esse operador é utilizado são designadas por expressões-$\lambda$. As expressões-$\lambda$ são expressões lógicas onde esse operador é utilizado para ligar as variáveis lógicas aos respectivos termos funcionais. Segundo Pereira e Shieber [PER87], as expressões-$\lambda$ podem aparecer onde qualquer símbolo de função (ou *functor*) seja igualmente permitido.

O operador-$\lambda$, proposto por Pereira e Shieber, é um operador infixo, com associatividade à direita, representado pelo símbolo ^. Numa expressão-$\lambda$, o símbolo ^ é utilizado como um operador de ligação. Por exemplo, a expressão $X \text{ ^ } (X+1)$ seria a expressão-$\lambda$ que representaria a função $f(x) = x+1$, em Prolog. Em notação lambda, essa função seria representada por uma expressão, como $\lambda x.(x+1)$. Nessa notação, uma expressão como $(\lambda x.\phi)a$ representaria a aplicação de uma função $f(x)$ a $a$, ou seja, $f(a)$. Nessa expressão, o termo $\phi$ representa uma expressão lógica, a expressão $\lambda x.\phi$ uma função $\mathrm{f}(x) = \phi$ e $x$ uma variável lógica. A aplicação $(\lambda x.\phi)a$ seria simplesmente a redução-$\beta$ dessa expressão, ou seja, a expressão equivalente $[\phi]\{x \mapsto a\}$ que resultaria da substituição $\{x \mapsto a\}$.

Pereira e Shieber [PER87] consideram a composição semântica como a aplicação de reduções-$\beta$ a expressões-$\lambda$ e sugerem, portanto, para esse efeito, a utilização de um predicado, como reduce($Arg$ ^ $Expr$, $Arg$, $Expr$). Por exemplo,

uma expressão como reduce($Function, a, Result$) seria interpretada como uma expressão do tipo $Result = [Expr]\{Arg \mapsto a\}$, onde $Function = Arg \wedge Expr$. A resolução do literal reduce($Function, a, Result$) seria precisamente a redução-$\beta$ da expressão $(Arg \wedge Expr)(a)$, ou seja, o resultado da aplicação da substituição, $\{Arg \mapsto a\}$, à expressão $Exp$, $[Expr]\{Arg \mapsto a\}$.

Mas como a resolução do literal $reduce/3$ é determinista e envolve apenas a ligação de variáveis lógicas, o literal pode ser dispensado e a ligação feita no próprio local onde são utilizadas. Por exemplo, um literal como reduce($A, B, C$) exige apenas que $A$ seja da forma $B \wedge C$. A variável $A$ pode, portanto, ser substituída por uma expressão-$\lambda$, como $B \wedge C$. De facto, a resolução do literal reduce($A, B, C$) seria uma expressão do tipo $A = B \wedge C$. Segundo Pereira e Shieber, essa técnica é conhecida por execução parcial.

Como veremos na Secção 4.2, a formação de expressões matemáticas é feita a partir da sua representação natural ou quasi-natural. As expressões matemáticas são analisadas gramaticalmente por regras DCG, como e_op_noun($Op, Arg1 \wedge Arg2 \wedge E$), e traduzidas para a sua representação lógica por composição semântica. A reescrita de expressões matemáticas é feita também por composição lógica de construtores-$\lambda$, utilizando uma técnica semelhante à que foi apresentada aqui.

## 3.5 A Reescrita de Expressões

A reescrita de expressões é uma parte importante da resolução simbólica. Segundo Meseguer [MES98], a lógica da reescrita exprime uma equivalência essencial entre lógica e computação. Um axioma da lógica de reescrita, escrito como $l \rightarrow r$, pode ser interpretado como uma regra de inferência. Mas o que se pretende com a reescrita de expressões, na maioria das vezes, segundo Dershowitz

et al. [DER01], é obter apenas formas equivalentes, mais simples, dessas expressões. Porém, como veremos, na Secção 4.3, nem sempre essas formas são mais simples. Todas elas, porém, contribuem para a resolução simbólica dessas expressões, através da formação de *redexes*.

Segundo Dershowitz et al. [DER01], a reescrita de expressões é um método bastante poderoso de lidar computacionalmente com equações. Através da reescrita, é possível substituir termos, numa expressão, por outros que lhes são equivalentes, e obter assim formas equivalentes dessas expressões. A ideia central por detrás da teoria de reescrita é a imposição de direccionalidade no uso de equações em demonstrações. A reescrita axiomática é baseada nesse princípio mas extensiva apenas à transformação lógica de expressões com estruturas triangulares bem definidas.

Como veremos na Secção 4.3.1 as regras de reescrita estão orientadas no sentido em que devem ser aplicadas. Nos casos em que a aplicação pode ser feita em ambos os sentidos, as regras estão orientadas em ambos os sentidos.

**Definição 3.5-1** ***Regra de Reescrita***

> *Uma regra de reescrita é um par ordenado de termos* $l$ *e* $r$*, escrito como* $l \to r$.

Segundo Dershowitz et al. [DER01], as regras de reescrita são formas orientadas de equações com as quais se podem efectuar substituições por unificação (Secção 3.3.2) dos seus lados esquerdos com expressões ou outras equações. Como uma forma de computação, as regras de reescrita são aplicadas, normalmente, de uma forma não determinista. Em princípio, numa reescrita de expressões, pode haver mais do que uma regra aplicar e mais de que uma posição onde aplicar essa regra.

A orientação das regras de reescrita deve, no entanto, obedecer a certos

requisitos fundamentais:

- ***Uma variável não pode aparecer isolada no lado esquerdo de uma regra.*** Por exemplo, uma regra como $x \to x + 0$, não é considerada computacionalmente válida;
- ***Uma variável não pode aparecer no lado direito de uma regra, sem que apareça também no seu lado esquerdo.*** Por exemplo, uma regra como $y + 1 \to x$, não é computacionalmente válida.

Segundo Dershowitz et al. [DER01], a teoria de reescrita está centrada no conceito de forma normal (Secção 2.5). O conceito de normalização está intimamente ligado ao conceito de redução-$\beta$. A presença, ou ausência de expressões redutíveis, ou *redexes*, é um indicador importante na reescrita de expressões. Valença e Barros [VAL00] consideram uma forma redutível de uma expressão como uma forma continuada da expressão. Podíamos dizer que a forma normal de uma expressão, $\mathcal{N}orm(s)$, é uma forma não continuada da expressão. Numa forma continuada, $\mathcal{R}edex(s)$, existem, portanto, um ou mais *redexes*.

Segundo Baader e Nipkow [BAA98], um sistema de reescrita $\mathcal{R}$ é convergente se for confluente e terminante. $\mathcal{R}$ é confluente se e só se, para $s,u,v \in \mathcal{T}$, $u \,{}^{*}\!\leftarrow s \to^{*} v$[32] implica que $u \downarrow v$, ou seja, $\mathcal{N}orm(u) = \mathcal{N}orm(v)$ [BAA98, DER01]. $\mathcal{R}$ é terminante se não existirem cadeias infinitas de redução.

**Exemplo 3.5-1** ***Convergência***

*Se* $\mathcal{T} = \{a, b, c\}$ *e* $\mathcal{R} = \{a \to b, a \to c, b \to c\}$, *então a aplicação* $\mathcal{R}(\mathcal{T})$ *conduz sempre ao termo* $c$. *Neste caso,* $c$ *é o termo mais simples de* $\mathcal{T}$,

[32] O símbolo $\to^{*}$ representa aqui o fecho reflexivo e transitivo de $\to$, isto é, $\to^{*} := \bigcup_{i \in \mathbb{N}} \to^{i}$.

*segundo* $\mathcal{R}$, *ou seja,* $c = \mathcal{N}orm(a) = \mathcal{N}orm(b)$. $\mathcal{R}$ *é, neste caso, convergente. Mas deixaria de o ser se adicionássemos a regra* $b \to a$ *a* $\mathcal{R}$. *Isto porque,* $a \to b, b \to a \in \mathcal{R}$.

---

A unicidade da forma normal depende, portanto, da completude do sistema. O sistema de reescrita é completo se for convergente.

A reescrita não é mais do que uma substituição de termos por outros que lhes são equivalentes. Se $s \in \mathcal{T}$ e $\sigma$ for uma substituição, então a transformada de $s$, $\sigma(s)$, é o termo $s\sigma$. Por convenção, a transformada é escrita em notação pós-fixa. A transformada de um termo é também designada por instância desse termo.

Se $s, t \in \mathcal{T}$, então diz-se que $s$ é uma variante alfabética de $t$, se e só se ambos forem instâncias um do outro, isto é, existirem duas substituições, $\sigma$ e $\omega$, tal que $t\sigma = s$ e $s\omega = t$ [DER01, STE00]. A renomeação de variáveis está intimamente ligada a esse conceito. Por exemplo, termos como $\log(e, \text{pwr}(Arg21, Arg22))$ e $\log(e, \text{pwr}(Exp, Base))$ são variantes alfabéticas do mesmo termo.

Dados dois termos $s, t \in \mathcal{T}$, se $s$ for um subtermo de $t$, então $t[s]$ representa $s$. Seja $\mathcal{R}$ um sistema de reescrita, convergente ou não. Se $s$ for uma instância $l\sigma$ do lado esquerdo de uma regra $l \to r \in \mathcal{R}$ [33], então $s$ é um $\mathcal{R}edex(s)$ em $t$, ou seja, um termo que pode ser reduzido[34] a uma instância $r\sigma$ do lado direito dessa regra. Diz-se então que $t[s] \to t[r\sigma]$, ou seja, que $t[s]$ pode ser reescrito como $t[r\sigma]$.

[33] O símbolo $\sigma$ é o unificador mais geral desse termo e o lado esquerdo da regra, isto é, $\sigma = \text{mgu}(s, l)$.

[34] Dershowitz et al. utiliza o termo "*contracted*", para designar o mesmo.

Diz-se que o subtermo $t[s]$ é o $\mathcal{R}edex(s)$, mais interior[35] de $t$, se $t[s]$ for um *redex* e nenhum subtermo próprio de $s$, $s[u]$, for também um *redex*. Por exemplo, a expressão $x+0$ em $(x+0)+x$ é um *redex* desse tipo. Diz-se também que o subtermo $t[s]$ é o $\mathcal{R}edex(s)$, mais exterior[36] de $t$, se $t[s]$ for um *redex* e $t$ não for um subtermo próprio $u[t]$ de $\mathcal{R}edex(u)$. Por exemplo, a expressão $(x+x)+0$ é um *redex* desse tipo. Nessa expressão, $x+x$ seria o *redex* mais interior.

Beeson [BEE98] e Dershowitz et al. [DER01] consideram que muitos dos axiomas em Matemática, são difíceis de lidar se por reescrita. Dershowitz et al. [DER01] consideram o axioma da comutatividade, $a+b=b+a$, como não terminável, seja qual for a orientação que se lhe dê como regra de reescrita. No caso dos axiomas de associatividade, $abc=(ab)c$ e $a+b+c=(a+b)+c$, seria necessário considerar todas as formas possíveis de associar e comutar os termos, pois todas elas seriam equivalentes. Segundo Dershowitz et al. [DER01], isso imporia um enorme fardo ao completamento do sistema de reescrita.

No caso da reescrita axiomática, esse problema não se coloca pois esses axiomas só são aplicados se contribuírem para a eliminação ou formação de novos *redexes* e não violarem o princípio de exclusão (Secção 4.4.2).

Dershowitz et al. [DER01] sugere, no caso da comutatividade, que essa propriedade seja apenas aplicada como forma de facilitar a aplicação de outras regras. Por exemplo, para aplicar uma regra, como $x+0\rightarrow x$, a $0+a$, bastaria apenas comutar os termos de $0+a$.

Muitos dos problemas que pretendemos resolver com a reescrita são, por

[35] Termo que, na literatura inglesa, é referido por "*innermost redex*"

[36] Tradução do termo "*outermost redex*"

natureza, condicionais e, portanto, necessitam de sistemas de reescrita condicional. Numa reescrita condicional, as regras de reescrita estão sujeitas a condições de aplicabilidade. Segundo Dershowitz et al. [DER01], uma regra de reescrita condicional é uma regra escrita como $C \mid l \to r$, onde $C$ é encarada como uma *guarda* ou condição de aplicabilidade dessa reescrita. As condições, ou *guardas*, podem assumir formas variadas, como por exemplo, expressões lógicas, equações ou inequações. Uma reescrita condicional só pode ter lugar se as condições de aplicabilidade da regra, com a qual o termo unifica, forem satisfeitas. Uma regra de reescrita sem *guardas* é uma regra de reescrita incondicional.

Numa regra de reescrita condicional, $C \mid l \to r$, as *guardas* $C$ são designadas por premissas e a reescrita, propriamente dita, $l \to r$, por conclusão. O significado que é, normalmente, atribuído à regra $C \mid l \to r$ é que $C \Rightarrow l \to r$ [37].

Segundo Dershowitz et al. [DER01], uma das propriedades mais importantes da reescrita é a propriedade de terminação. Quando um sistema de reescrita é aplicado a um termo, é importante que o processo de reescrita termine eventualmente, independentemente, da forma ou modo, como as suas regras são aplicadas. Se um sistema é terminável (ou normalizável), então é sempre possível encontrar uma forma normal de um termo, por uma qualquer sequência de reescrita, desde que essa reescrita continue pelo tempo que for necessário. Essa forma pode não ser a única.

Seja $\mathcal{T}$ um conjunto de termos e $\to$ uma relação binária sobre $\mathcal{T}$. Segundo Dershowitz et al. [DER01], a relação $\to$ é terminável para $\mathcal{T}$, se para qualquer termo $t_0 \in \mathcal{T}$, não existir nenhuma cadeia de derivação infinita, $t_0 \to t_1 \to t_2 \to \cdots$ de termos $t_i \in \mathcal{T}$.

[37] O símbolo $\Rightarrow$ é utilizado aqui como implicação.

Dershowitz et al. [DER01] consideram que devem apenas ser consideradas derivações onde nenhum *redex* seja deixado por reduzir por tempo indefinido. Para além da terminação (ou normalização), os sistemas de reescrita $\mathcal{R}$ devem usufruir também a propriedade de confluência.

Seja $\mathcal{T}$ um conjunto de termos e $\rightarrow$ uma relação binária sobre $\mathcal{T}$. Segundo Baader [BAA98] e Dershowitz et al. [DER01], a relação $\rightarrow$ é confluente, se existir um elemento $v \in \mathcal{T}$, tal que, $s \rightarrow^* v$ e $t \rightarrow^* v$, sempre que $u \rightarrow^* s$ e $u \rightarrow^* t$, para alguns elementos $s,t,u \in \mathcal{T}$. Uma relação confluente possui também a propriedade de Church-Rosser. Relações confluentes e termináveis (ou normalizáveis) são designadas convergentes. Apenas as relações convergentes garantem a unicidade da normalização. Um sistema de rescrita $\mathcal{R}$ é convergente se a sua relação de reescrita também o for.

A resolução simbólica de expressões é um sistema de reescrita convergente que depende apenas da completude da sua base axiomática.

## 3.6 Conclusões

Neste Capítulo foi apresentado o trabalho relacionado e o enquadramento teórico em que o nosso trabalho de investigação se baseou. O trabalho de investigação, apresentado nesta tese, centrou-se essencialmente no desenvolvimento de uma linguagem computacional para a resolução simbólica de expressões matemáticas, baseada na representação lógica de expressões matemáticas como estruturas triangulares bem definidas, e na reescrita axiomática como método computacional.

O conceito de estrutura triangular foi abordado no Capítulo 2 e a noção de reescrita axiomática com base nesse tipo de estrutura será abordado no Capítulo 4.

Esses dois conceitos são os principais contributos desta tese.

O Assistente de Matemática, que será apresentado no Capítulo 4, é uma aplicação prática destes dois conceitos.

# Capítulo 4 O Trabalho de Investigação

## UMA LINGUAGEM COMPUTACIONAL PARA A RESOLUÇÃO SIMBÓLICA DE EXPRESSÕES

## 4.1 Introdução

Nas palavras de Ravaglia et al. [RAV98],

> **"In designing these courses [EPGY courses] we have tried to be as responsive as possible to variations in student abilities and rates of learning. Because of the opportunities for assessment it affords, symbolic computation has been an essential tool in the instructional design of these courses."**[38]

a computação simbólica foi uma ferramenta essencial na concepção dos cursos de instrução programada para o programa EPGY. Segundo Ravaglia et al. [RAV98], a computação simbólica oferece enormes oportunidades para a avaliação contínua e sistemática das capacidades de raciocínio dos alunos. Uma das áreas onde a computação simbólica pode ser especialmente útil é, precisamente, na área relacionada com a simplificação de expressões matemáticas.

[38] "*Na concepção desses cursos [cursos para o programa EPGY], tentámos responder o melhor possível às variações de capacidade dos alunos e às suas taxas de aprendizagem. Devido às oportunidades para avaliação que ela proporciona, a computação simbólica tem sido uma ferramenta essencial na concepção do modelo instrucional para esses cursos.*" (Tradução)

## 4.2 A Formação de Expressões

O conceito de expressão matemática, que é defendido nesta tese, envolve noções tão simples como constantes, variáveis e operadores. Todas essas noções estão intimamente ligadas à noção de estrutura triangular, que definimos no Capítulo 2.

As expressões matemáticas, incluídas nesta tese, são tratadas, simbolicamente, como estruturas triangulares, bem definidas. Como veremos, esse tipo de estrutura é, facilmente, manipulável por linguagens de programação lógica, como o Prolog. Por exemplo, uma expressão matemática, como $\log_e 4^2$, pode ser representada, em Prolog, por uma expressão lógica do tipo $\log(e, \text{pwr}(2,4))$. Como vimos no Capítulo 2, uma expressão lógica deste tipo possui uma estrutura triangular do tipo *log_pwr*.

Veremos também que, com este tipo de estrutura, é possível simplificar expressões, de uma forma simbólica, por reescrita axiomática.

Podemos dizer que

> **A reescrita axiomática não é mais do que a aplicação de uma regra matemática, por reescrita.**

De facto, o resultado de uma reescrita axiomática é uma expressão equivalente que resulta da substituição de um dos seus termos por uma instância do lado direito da regra que lhe foi aplicada.

Ao processo de simplificação por reescrita axiomática demos o nome de resolução simbólica de expressões matemáticas por reescrita axiomática. As expressões matemáticas incluídas nesta tese são formadas pela conjunção lógica de um, ou mais, termos Prolog. Ao processo de formação demos o nome de composição lógica por construção-$\lambda$.

### *4.2.1 A Representação Lógica do Sinal*

O símbolo ‘ – ’ pode representar também o *sinal negativo* de uma expressão. Expressões como $-\log_e 4^2$ são expressões que designamos por simétricas. Como veremos, mais adiante, o sinal negativo é também representado por uma expressão lógica. Essa forma de representar o sinal [COH03] tornou possível a representação lógica de expressões como $-x$ e $4+(-1)x$ . O sinal possui uma estrutura triangular do tipo $\Xi$-*prod* .

Operações simbólicas como, por exemplo, “*pôr o sinal em evidência*”, ou “*mudar o sinal*” lidam, essencialmente, com essa parte da expressão. Para representar expressões como $-x$, $-\sqrt{x}$ ou $-(x+1)$ é necessário pôr em evidência o sinal . O significado operacional que está, normalmente, associado ao sinal negativo, é o de um produto envolvendo a constante $-1$.

O elemento $-1$ é representado pelo termo atómico, $-1$, ou por um termo funcional do tipo $\text{prod}(sign,1)$ definido recursivamente como

1. $sign = -1$ ; ou
2. $sign = \text{prod}(sign,1)$

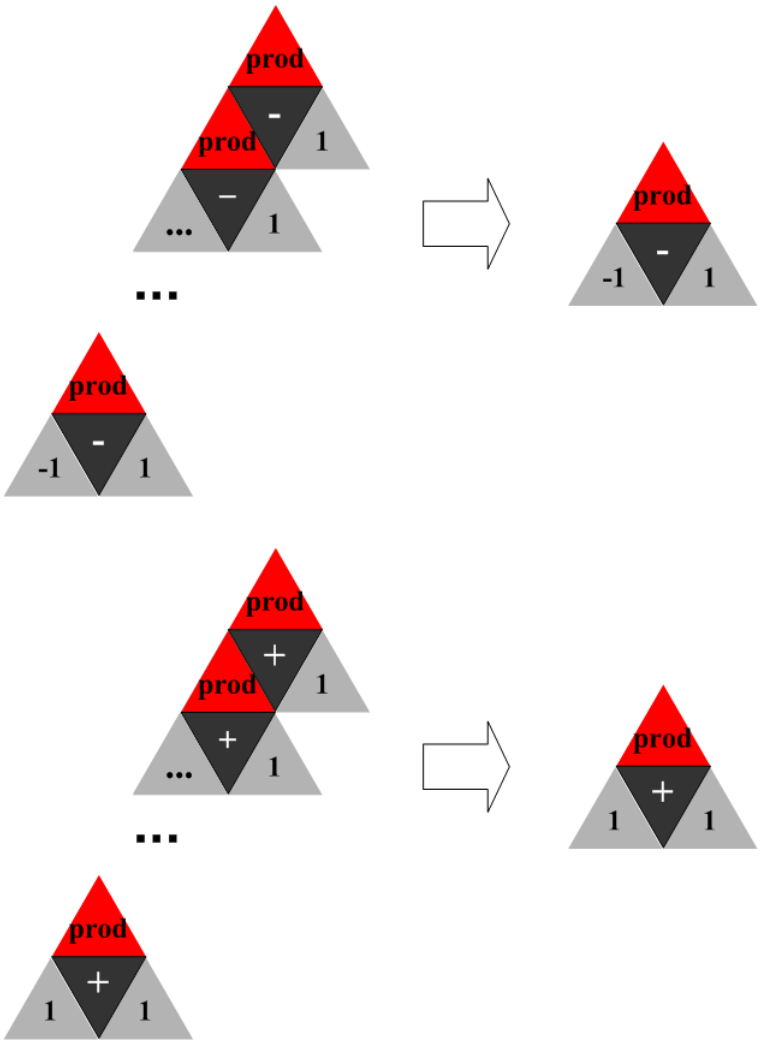



**Figura 4.2.1** *Estrutura do sinal*

A presença de qualquer um desses termos, $sign$ ou $\text{prod}(sign,1)$, como factor mais à esquerda, é interpretada como o sinal negativo da expressão. O termo funcional $\text{prod}(sign,1)$ é redutível a $sign$. Por simetria, a presença da constante $1$, como factor mais à esquerda, é interpretado como sinal positivo. Uma estrutura como $\text{prod}(sign, sign)$, é redutível a $1$, ou seja,

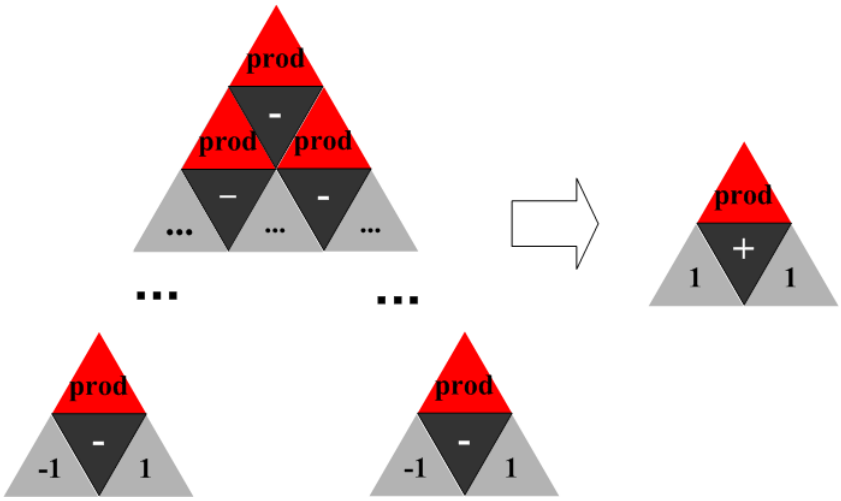



**Figura 4.2.2** ***Composição lógica de sinais***

Os números inteiros negativos, como por exemplo $-3$, são representados por termos funcionais, como $\text{prod}(-1,3)$. Esta representação confere um significado operacional ao símbolo "$-$". A representação do sinal como expressão lógica foi extremamente útil para o tratamento do sinal de expressões não numéricas.

O sinal negativo é lido e interpretado como "*menos*" ou "*a simétrica de*". A forma como a expressão é lida determina também a forma como o sinal é representado.

**Exemplo 4.2-1** ***Representação lógica de*** $-(-2)$

*A expressão* $-(-2)$, *lida como "menos menos 2", "menos* $-2$*", "a simétrica da simétrica de 2" ou "a simétrica de* $-2$*", é representada por* $\text{prod}(-1,\text{prod}(-1,2))$. *Lida como "o produto de* $-1$ *por* $-2$*", seria representada por* $\text{prod}(\text{prod}(-1,1),\text{prod}(-1,2))$.

A resolução simbólica de uma expressão reduz sempre o sinal à sua forma mais simples (isto é, à sua forma normal e única). No caso do Exemplo 4.2-1, essa expressão seria a constante numérica 2. O tratamento seria o mesmo, independentemente do número de sinais que pudessem estar presentes.

A base axiomática $\mathcal{B}(s)$, para esta resolução, seria, neste caso, completa (isto é, a resolução seria convergente). A sua classe de reescrita seria $\mathcal{R}w = \{prod, prod\_prod\}$. A estrutura *prod* tem, neste caso, um poder redutor mais elevado. Note que nesta classe não está incluída a assinatura *prod_prod_prod*. Podíamos ter definido uma regra com essa assinatura mas não foi necessário. Como se sabe, uma estrutura Ξ-*prod_prod_prod* pode ser tratada como duas estruturas Ξ-*prod_prod*.

As regras a aplicar, neste caso, seriam as seguintes:

**R1.** $\text{prod}(s,t) \to \text{prod}(q,r)$ **if** $s,t \in \mathcal{T}^- \wedge q = |s| \wedge r = |t|$

**R2.** $\text{prod}(s,t) \to r$ **if** $s,t \in \mathbb{N} \wedge r = s \cdot t$

Mas num cálculo simbólico, nem sempre é conhecido o sinal da expressão. Por exemplo, expressões como $-x$, representam apenas expressões simétricas de $x$. O sinal de $x$ é desconhecido. O sinal de $x$ depende portanto do valor que $x$ possa vir a assumir num determinado contexto.

Assim, consideramos que

> **a *variável matemática* assume apenas os valores que supostamente lhe podem ser atribuídos no contexto em que está inserida.**

Por exemplo, no caso de uma expressão, como $\sqrt{x}$, é assumido que $x \geq 0$, ou seja, que $\sqrt{x}$ representa um valor válido. Só assim se pode justificar a sua

inserção num contexto como $\log_e \sqrt{x}$. O valor $\sqrt{-2}$ seria considerado inválido, pois $-2$ é uma constante e, portanto, o seu valor é conhecido.

Consideramos que

> **O *sinal* de uma constante só é conhecido se a constante for irredutível.**

De facto, para determinar o sinal de uma constante simbólica, pode ser necessário resolver a constante simbolicamente. Uma constante inteira é irredutível e, portanto, o seu sinal é conhecido. Por exemplo, $-5 < 0$. Mas no caso de uma constante, como $-(-5)$, é necessário resolvê-la, pois só depois de resolvida é que se sabe que $-(-5) \to 5$. No caso de uma constante, como $-\log_e \frac{1}{2}$, teríamos

$$-\log_e \frac{1}{2} \to_{\Gamma_1} -(\log_e 1 - \log_e 2) \to_{\Gamma_2} -(0 - \log_e 2) \to_{\Gamma_3}$$

$$\to_{\Gamma_3} -(-\log_e 2) \to_{\Gamma_4} \log_e 2 > 0$$

onde símbolo $\Gamma_i$ representa o tipo de transformação lógica ou reescrita axiomática (redução-$\kappa$, conversão-$\alpha$, etc.) efectuada.

As regras aplicadas, neste caso, foram as seguintes:

**R1.** $\log(s, \mathrm{div}(t,r)) \to \mathrm{diff}(\log(s,t), \log(s,r))$

**if** $c \wedge t \in \mathcal{K}^+ \setminus \{-1,0\} \wedge t \in \mathcal{K}^+ \setminus \{-1,0,1\} \wedge$

$\neg reducible(\mathrm{div}(t,r))$

**R2.** $\log(s,t) \to 0$ **if** $s \in \mathbb{N} \setminus \{0,1\} \cup \{e\} \wedge t \in \{1\}$

**R3.** $\text{diff}(s,t) \to \text{prod}(-1,t)$ **if** $s \in \{0\} \wedge t \notin \mathbb{N}$

**R4.** $\text{prod}(s,t) \to 1$ **if** $\textit{negative}(s) \wedge \textit{negative}(t)$

**R5.** $\text{prod}(s,t) \to t$ **if** $s \in \{1\} \wedge t \notin \mathbb{N}$

onde $\mathcal{K}^-$ é o conjunto de constantes simbólicas negativas e $\mathcal{K}^+ = \mathcal{K} - \mathcal{K}^-$, o conjunto das constantes simbólicas positivas.

Consideramos que

> **a *simetria* de uma expressão só pode ser determinada pela presença do sinal**

De facto, independentemente do sinal que a expressão possa vir a assumir, as expressões $-\log_e \frac{1}{2}$ e $\log_e \frac{1}{2}$ diferem apenas no sinal. Mas nem sempre o sinal de uma expressão está evidenciado dessa forma, ou seja, à esquerda do operador produto. Quando tal acontece, é necessário pô-lo em evidência. Por exemplo, a expressão $2(-x)$ não é considerada a simétrica de $2x$. Só o é depois de transformada em $-2x$. O resultado seria a seguinte transformação lógica

$$\underbrace{\text{prod}(2,\text{prod}(-1,x))}_{2(-x)} \to_\Gamma \underbrace{\text{prod}(-1,\text{prod}(2,x))}_{-(2x)}$$

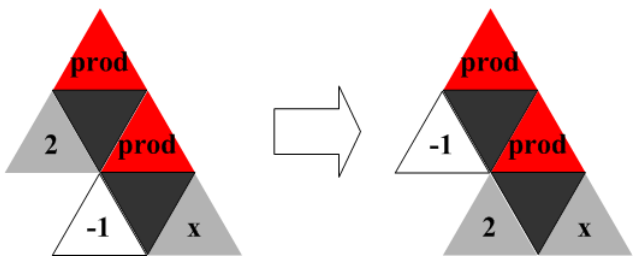



**Figura 4.2.3** *Colocar o sinal em evidência*

Note que ambas as estruturas são do mesmo nível $\mathrm{N2}$, mas a segunda irredutível. Podemos dizer que esse tipo de transformação diminui o poder redutor da estrutura. De facto, com o sinal colocado o mais à esquerda possível, essa parte da estrutura torna-se irredutível. Este exemplo mostra a importância que certas propriedades, como, por exemplo, a comutatividade e a associatividade, podem ter no tratamento simbólico de expressões.

A transformação $\Gamma$, ilustrada na Figura 4.2.3, é precisamente o resultado da aplicação da propriedade associativa do produto. Colocar o sinal em evidência pode ser também o resultado da aplicação da propriedade distributiva do produto. Por exemplo, $-x+1 \rightarrow -(x-1)$. Mas aqui, também, o nível permanece o mesmo. A razão para isso, em ambos os casos, está precisamente no facto desse tipo de transformação ser apenas o resultado de uma mudança da estrutura do sinal.

**Definição 4.2-1** ***Sinal Negativo***

> *O sinal negativo é um termo* $s \in \mathcal{S}ign$*, representado pelo termo atómico* $-1$*, ou por um termo funcional do tipo* $\mathrm{prod}(sign,1)$*, onde* $sign \in \mathcal{S}ign$*. O sinal negativo é um elemento do conjunto* $\mathcal{S}ign = \{-1\} \bigcup \mathcal{S}^{-1}$ *onde*
>
> $$\mathcal{S}^{-} = \{ f(s,t) \in \mathcal{F} \mid f \in \{prod\} \wedge s \in \mathcal{S}ign \wedge t \in \{1\} \}$$

O sinal negativo é uma estrutura triangular do tipo $(\mathrm{N1})_{prod}$ ou uma que é redutível a uma estrutura desse tipo. A Figura 4.2.4 mostra a estrutura triangular do sinal negativo.

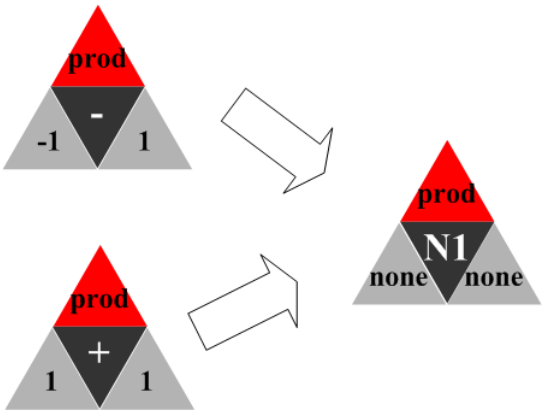



**Figura 4.2.4** *Estrutura triangular do sinal*

Como se pode observar, o sinal é uma expressão do tipo $\text{prod}(-1,1)$ ou $\text{prod}(1,1)$, isto é, uma expressão com uma estrutura do tipo $\Xi$-*prod*. Estruturas do tipo $\text{prod}(sign,1)$, onde $sign \in \mathcal{S}ign$, são redutíveis a esse tipo.

O que distingue o sinal negativo de outras estruturas do mesmo tipo, como por exemplo $2(3)$, é precisamente o facto do seu primeiro argumento ser $-1$. Uma estrutura, como $\text{prod}(-1,1)$, é irredutível. Uma expressão, como $2(3)$, possui uma estrutura do mesmo tipo, $\text{prod}(2,3)$, mas redutível. A presença de números inteiros negativos em expressões, como $-5(2)$, torna essas expressões mais complexas e portanto redutoras.

Nesta tese, todos os números inteiros negativos $i \in \mathcal{I}^-$, com excepção de $-1$, possuem estruturas do tipo $\Xi$-*prod*. O inteiro $-1 \in \mathcal{A}$ é considerado uma constante especial e utilizado apenas como representação lógica do sinal na estrutura $\text{prod}(sign,\_)$. O números naturais $i \in \mathbb{N}$, possuem estruturas do tipo $\Xi$-*none*.

**Definição 4.2-2** ***Expressão Simétrica***

> *Uma expressão simétrica* $s \in \mathcal{T}^-$ *é uma expressão lógica do tipo*

> $\text{prod}(sign, \_)$, *onde* $sign \in \mathcal{S}^-$, *representa o sinal negativo e* $\_$, *uma expressão lógica* $t \in \mathcal{T}$. *Uma expressão simétrica é um elemento do conjunto*
>
> $$\mathcal{T}^- = \left\{ f(s,t) \in \mathcal{F} \mid f \in \{prod\} \wedge s \in \mathcal{S}^- \wedge t \in \mathcal{T} \right\}$$

Expressões, como $-(-(-(6)))$ ou $-2\sqrt{2}$, representam expressões simétricas se o sinal mais à esquerda estiver em evidência. Como já foi mencionado atrás, a forma como a expressão é lida determina a sua estrutura.

Nesta tese, optámos por considerar como simétricas, apenas as expressões com estruturas do tipo $\text{prod}(sign, \_)$, cujo sinal é negativo. De facto, uma expressão $s$ só pode ser considerada simétrica de uma com uma estrutura do tipo $\text{prod}(sign, s)$.

### *4.2.2 A Composição Lógica por Construção -λ*

As expressões matemáticas, incluídas nesta tese, são formadas por composição lógica de um, ou mais, termos. Termos que designamos por construtores -$\lambda$. A designação provém do facto de termos optado por um método de composição semelhante ao proposto por Pereira e Shieber [PER87]. A composição lógica por construção -$\lambda$ é utilizada na reescrita de expressões lógicas.

Nesta tese, as expressões matemáticas são formadas por escrita ou reescrita, ou seja, formadas a partir da sua representação natural (ou quasi-natural) ou como resultado de uma transformação lógica.

**Exemplo 4.2-2** ***Escrita de*** $\log_e 4^2$

> *A expressão* $\log_e 4^2$, *cuja estrutura é do tipo* $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$, *é escrita por composição lógica de expressões -*$\lambda$. *Essa escrita é o resultado da*

*análise sintáctica e semântica da frase*

*"definir a expressão log e pwr 2 4"*

*As regras DCG aplicadas, neste caso, seriam as seguintes:*

$$s(S) \rightarrow \text{define\_verb}(Arg \text{ ^ } S), \text{expression}(Op, Arg).$$

$$\text{define\_verb}(Arg \text{ ^ } \text{define}(\text{practice}(\text{expression}(Arg)))) \rightarrow [Op], \{\text{verb}(define, Op)\}.$$

$$\text{expression}(Op, Arg) \rightarrow \text{op\_noun}(Op, Arg1 \text{ ^ } Arg2 \text{ ^ } Arg), \text{expression}(Op1, Arg1), \text{expression}(Op2, Arg2).$$

$$\text{op\_noun}(\log, Arg1 \text{ ^ } Arg2 \text{ ^ } \log(Arg1, Arg2)) \rightarrow [Op], \{\text{operator}(\log, Op)\}.$$

$$\text{op\_noun}(\text{pwr}, Arg1 \text{ ^ } Arg2 \text{ ^ } \text{pwr}(Arg1, Arg2)) \rightarrow [Op], \{\text{operator}(\text{pwr}, Op)\}.$$

Para mais informação sobre a gramática DCG pode consultar [PER87].

**Exemplo 4.2-3** ***Reescrita de*** $\log_e 4^2$

*A expressão* $\log_e 4^2$*, cuja estrutura é do tipo* $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$*, é reescrita por composição lógica de dois (2) construtores-*$\lambda$*: um, para a formação de logaritmos,* $\log(e, Pwr, Log)$*, e outro, para a formação de potências,* $\text{pwr}(2, 4, Pwr)$*. A variável* $Pwr$ *é a variável de ligação. A resolvente (Secção 3.3.2) dessa composição é o resultado da composição das duas substituições seguintes*

$$\sigma_{pwr} = \{Arg1 \mapsto 2, Arg2 \mapsto 4, Pwr \mapsto \text{pwr}(2,4)\}$$

$$\sigma_{log} = \{Arg1 \mapsto e, Pwr \mapsto \text{pwr}(2,4), Log \mapsto \log(e, \text{pwr}(2,4)\}\ .$$

*A expressão* $\log_e 4^2$ *é simplesmente uma instância da variável* $Log$ *.*

---

**Definição 4.2-3** $Construtor\text{-}\lambda$

> *Um construtor-*$\lambda$ *, denotado por* $\lambda\text{-}op$ *, é uma variante alfabética do predicado* $op(Arg_1, Arg_2, Op)$ *, onde* $op \in \mathcal{O}p$ *representa um tipo funcional e* $Op, Arg_1, Arg_2 \in \mathcal{V}$ *, são variáveis lógicas.*

Os construtores-$\lambda$ são regras de formação de operadores. Numa composição lógica, envolvendo $n$ operadores, são necessários $n$ construtores-$\lambda$ e $n-1$ variáveis de ligação. Todas as expressões matemáticas, incluídas nesta tese, são formadas por composição lógica, quer por escrita a partir da sua representação natural (ou quasi-natural), quer por reescrita axiomática.

A composição lógica de expressões é uma componente importante da formação de expressões. No caso de reescritas, as expressões são formadas a partir das regras de reescrita $\lambda\text{-}op$, $op(Arg1, Arg2, op(Arg1, Arg2))$. No caso de escritas, a formação é feita através de gramáticas lógicas (DCG), associando uma forma lógica a cada componente da expressão através de uma expressão-$\lambda$, $Arg1\ \hat{}\ Arg2\ \hat{}\ op(Arg1, Arg2)$. Existe, de facto, uma grande semelhança entre essas duas formas de lidar com a composição lógica de expressões.

As regras de formação de operadores $\lambda$-$op$ (ou as expressões-$\lambda$), definem aquilo que poderíamos designar por construtores de classes (ou tipos) de operadores. As expressões lógicas são apenas instâncias dessas classes.

**Exemplo 4.2-4** ***Classe*** *log*

*Uma expressão lógica, como* $\log(e,4)$*, pode ser considerada como uma instância da classe* $\log$*, ou seja, uma instância do termo funcional* $\log(Arg_1, Arg_2)$ *que define essa classe. Qualquer expressão, com uma estrutura do tipo* $\log(\_,\_)$*, pode ser considerada também um membro dessa classe.*

---

A instanciação de uma classe é feita por composição lógica e envolve a instanciação, por unificação (Secção 3.3.2), de todas as componentes que constituem essa classe. Por exemplo, o termo funcional $\log(Arg_1, Arg_2)$ define aquilo que poderíamos, normalmente, designar por classe de logaritmos. O seu construtor-$\lambda$, $\lambda$-*log*, é uma variante alfabética do termo $\log(Arg_1, Arg_2, Log)$. A correspondente expressão-$\lambda$ seria o termo $Arg_1 \wedge Arg_2 \wedge \log(Arg_1, Arg_2)$.

Uma subclasse, por outro lado, é definida pela composição lógica de dois, ou mais, construtores-$\lambda$ (ou expressões-$\lambda$). A composição lógica de dois construtores-$\lambda$ como, por exemplo, $\log(Arg_{11}, Pwr, Log)$ e $\text{pwr}(Arg_{21}, Arg_{22}, Pwr)$, ou de duas expressões-$\lambda$ como, por exemplo, $Arg_{11} \wedge Arg_{12} \wedge \log(Arg_{11}, Arg_{12})$ e $Arg_{21} \wedge Arg_{22} \wedge \text{pwr}(Arg_{21}, Arg_{22})$), definiria aquilo que poderíamos designar, normalmente, por uma subclasse de logaritmos, cujos argumentos, neste caso, seriam potências, ou seja, uma subclasse *log_pwr*. Essa subclasse seria definida por um termo composto do tipo $\log(Arg_{11}, pwr(Arg_{21}, Arg_{22}))$. Uma expressão lógica, do tipo $\log(\square, \text{pwr})$, seria uma instância dessa classe. Uma expressão, como $\log_e 4^2$, seria considerada uma instância dessa classe.

Verificámos, também, que todas as expressões matemáticas, estudadas para esta tese, podem ser caracterizadas pela estrutura triangular da sua componente principal. As componentes, por serem triangulares, envolvem, no máximo, três operadores matemáticos. Como sabemos, esses operadores definem a assinatura dessas componentes. A assinatura de uma expressão é, portanto, a mesma que a da sua componente principal. A caracterização de uma expressão através da sua assinatura permite que uma reescrita axiomática pode ser efectuada ao nível da sua componente principal e, portanto, independentemente do contexto onde possa estar inserida ou dos argumentos para os quais sirva de contexto.

Consideramos, portanto, que

> **a assinatura é uma forma bastante eficaz de unificar expressões e regras matemáticas.**

De facto, a assinatura de uma expressão garante que só possam ser-lhe aplicadas regras com a mesma estrutura de classe. A reescrita de uma expressão matemática depende portanto da sua assinatura e da respectiva base axiomática. Note que cada componente é, de facto, a componente principal de uma outra expressão. Para se efectuar uma reescrita é necessário que na base axiomática existam regras para a sua estrutura de classe.

**Exemplo 4.2-5** ***Base axiomática para a estrutura*** $\Xi$-*log_pwr*

*Uma expressão, como* $\log_e 4^2$, *com uma assinatura* $log\_pwr$, *só pode ser reescrita por regras com assinaturas* $log$ *e* $log\_pwr$.

*A sua base axiomática é o conjunto*

$$\mathcal{B}_{log\_pwr} = \left\{ l_i \to r_j \in \mathcal{B} \mid i \neq j \wedge i, j \in \mathcal{R}w_{log\_pwr} \right\}$$

*onde*

$$\mathcal{R}w_{log\_pwr} = \{log, log\_pwr, prod, prod\_log\}$$

*é a sua classe de reescrita.*

---

Como vimos na Secção 2.4.1, a estrutura de uma expressão matemática, com as características definidas nesta tese, pode ser totalmente descrita a partir de cinco estruturas triangulares básicas. A estrutura de uma expressão é o resultado de uma composição lógica envolvendo um número limitado de estruturas triangulares básicas. Cada uma dessas estruturas possui uma assinatura *sig* e uma base axiomática $\mathcal{B}_{sig}$, bem definidas.

Expressões com uma estrutura básica Ξ-*none* são consideradas irredutíveis. Expressões, como 4 e $x$, são expressões desse tipo. Todas as outras estruturas básicas podem ser, ou não, redutíveis. Se forem redutíveis, então as respectivas estruturas só podem ser redutoras ou potencialmente redutoras. Uma expressão matemática pode ser decomposta num número finito de estruturas básicas e resolvida simbolicamente componente a componente. Uma estrutura do tipo [Ξ-*log_pwr*][Ξ-*pwr*], por exemplo, pode ser decomposta em duas outras, Ξ-*log_pwr* e Ξ-*pwr* e resolvida, simbolicamente, como o resultado da composição de duas outras resoluções.

Podemos, portanto, dizer que

> **a resolução simbólica de expressões é, simplesmente, o resultado da composição de outras resoluções.**

As estruturas irredutíveis não geram novas estruturas e, portanto, não contribuem para a proliferação de estruturas que podem ou não ser decompostas ou transformadas.

Por exemplo, a reescrita de uma componente Ξ-*log_pwr* daria origem a uma nova estrutura, Ξ-*prod_log*. As respectivas estruturas de classe seriam as seguintes

***Log_pwr***

$$\mathcal{S}ig_{log\_pwr} = \{log, log\_pwr\}$$

$$\mathcal{TS}ig_{log\_pwr} = \{prod, prod\_log\}$$

***Prod_log***

$$\mathcal{S}ig_{prod\_log} = \{prod, prod\_log\}$$

$$\mathcal{TS}ig_{prod\_log} = \{log, log\_pwr\}$$

A classe de reescrita da componente Ξ-*log_pwr*, seria

$$\mathcal{R}w_{log\_pwr} = \{log, log\_pwr, prod, prod\_log\}$$

A base axiomática para essa estrutura seria a união de duas bases, ou seja,

$$\mathcal{B} = \mathcal{B}_{log\_pwr} \cup \mathcal{B}_{pwr}$$

Essa seria, por exemplo, a base axiomática para a resolução simbólica de uma expressão, como $\log_e 4^2$.

A base axiomática para cada classe está organizada, hierarquicamente, de acordo com a sua estrutura de classe. A questão que se coloca é saber se a base axiomática de uma resolução, ou seja, $\mathcal{B}(s) = \bigcup_{i \in \mathcal{S}ig(s)} \mathcal{B}_i$, é completa, onde

$$\mathcal{S}ig(s) = \{ i \in \mathcal{S}ig \mid n \in \mathbb{N} \wedge i = signature(s,n) \}$$

$$\mathcal{B}_{sig} = \{ l_i \to r_j \mid i \neq j \wedge i, j \in \mathcal{R}w_{sig} \}$$

e $i = signature(s,n)$ é uma função que retorna a assinatura $i$ da componente $n$ de $s$.

Segundo Baader e Nipkow [BAA98], uma relação de reescrita $\to$ é convergente se e só se for confluente e terminante. No caso da resolução simbólica, essa relação é estritamente axiomática, isto é, a rescrita é o resultado da aplicação de axiomas e teoremas elementares. As regras de reescrita representam fórmulas matemáticas com assinaturas bem definidas e todas essas regras geram assinaturas para as quais existem regras de reescrita, redutoras ou potencialmente redutoras. Como sabemos, a classe de reescrita de uma expressão é, por definição, o conjunto de todas as assinaturas relacionadas com a sua estrutura de classe. A base axiomática de uma resolução é completa se existirem regras de reescrita para todas as assinaturas na sua classe de reescrita.

Consideramos, portanto, que

> **uma resolução simbólica é confluente e terminante se a sua base axiomática for completa, isto é, existirem regras de reescrita com assinaturas que sejam membros da sua classe de reescrita.**

O princípio de exclusão garante a unicidade de todas as formas equivalentes de uma expressão, obtidas por reescrita axiomática, e portanto, da sua forma mais simples, também. A base axiomática para cada classe de rescrita é completa.

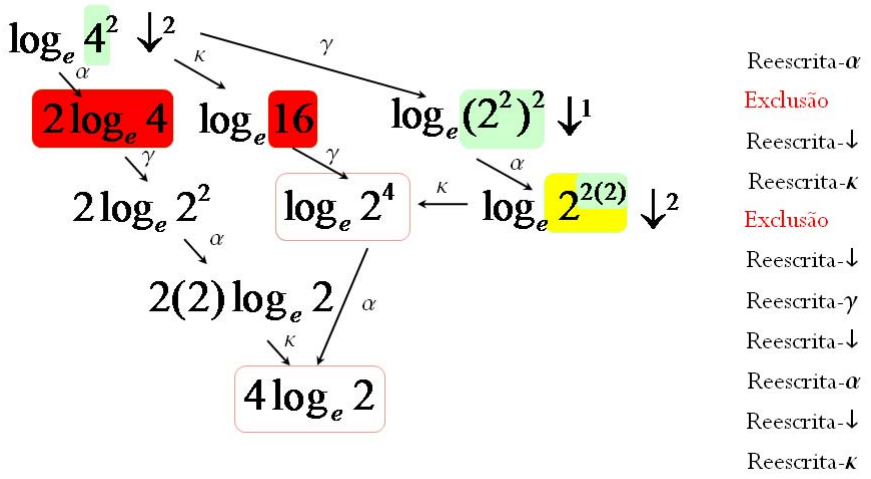
**Exemplo 4.2-6** ***Base de resolução de*** $\log_e 4^2$

*No caso da expressão* $\log_e 4^2$*, a base de resolução é convergente. Essa base é definida por*

$$\mathcal{B}(s) = \left\{ l_i \to r_j \in \mathcal{B} \mid i \neq j \wedge i, j \in \mathcal{R}w(s) \right\}$$

$$\mathcal{R}w(s) = \left\{ prod, prod\_log, pwr, pwr\_pwr, log, log\_pwr \right\}$$

*Todas as regras em* $\mathcal{B}(s)$ *conduzem a expressão à sua única forma normal, ou seja, à sua forma mais simples.*




**Exemplo 4.2-7** ***Base de resolução de*** $\log_{10} 5^3 + 4\log_{10} 2$

*No caso da expressão* $\log_{10} 5^3 + 4\log_{10} 2$, *a base axiomática é também convergente. Neste caso estão envolvidas mais assinaturas. A expressão possui uma estrutura triangular do tipo*

$$[\Xi\text{-}sum\_log\_prod][\Xi\text{-}log\_pwr][\Xi\text{-}prod\_log][\Xi\text{-}pwr][\Xi\text{-}prod]$$

*A base de resolução, neste caso, é definida por*

$$\mathcal{B}(s) = \left\{ l_i \to r_j \in \mathcal{B} \mid i \neq j \wedge i, j \in \mathcal{R}w(s) \right\}$$

*onde*

$$\mathcal{R}w(s) = \{ sum, sum\_log\_prod, sum\_log\_log,\ prod, prod\_log,$$

$$log, log\_pwr, log\_prod,\ pwr, pwr\_pwr \}$$

*é a sua classe de reescrita. A sua forma mais simples é a expressão* $\log_{10} 2+3$.

*A sua classe de reescrita é a união das classes de reescrita de todas as suas componentes, ou seja,*

$$\mathcal{R}w(s) = \mathcal{R}w_{sum\_log\_prod} \cup \mathcal{R}w_{log\_pwr} \cup \mathcal{R}w_{prod\_log} \cup \mathcal{R}w_{pwr} \cup \mathcal{R}w_{log}$$

*onde*

$$\mathcal{R}w_{sum\_log\_prod} = \{ sum, sum\_log\_log, prod, log, log\_prod \}$$

$$\mathcal{R}w_{log\_pwr} = \{ log, log\_pwr, prod, prod\_log \}$$

$$\mathcal{R}w_{prod\_log} = \{ log, log\_pwr, prod, prod\_log \}$$

$$\mathcal{R}w_{pwr} = \{ pwr, pwr\_pwr \}$$

---

Um dos objectivos desta tese foi, precisamente, o desenvolvimento de uma estrutura computacionalmente eficiente que pudesse facilitar a resolução simbólica, por reescrita axiomática, de uma certa classe de expressões matemáticas.

Como vimos, a classe de expressões, estudadas para esta tese, pode ser representada por estruturas triangulares e resolvida, simbolicamente, de uma forma detalhada, com base nessa estrutura. Essa classe foi caracterizada na Secção 2.4.

Numa reescrita axiomática, com fins estritamente pedagógicos, só devem ser incluídas transformações lógicas que sejam pedagogicamente justificáveis, isto é, cuja aplicação possa ser justificada em termos de um conjunto restrito de axiomas e teoremas, considerados elementares (isto é, transformações lógicas que Beeson [BEE98] designa por "*cognitive faithful*").

Nesta tese, só foram incluídas expressões matemáticas cujas componentes possuam estruturas triangulares com assinaturas que foram listadas na Secção 2.4. Embora os polinómios possam, em princípio, ser representados por estruturas triangulares, o seu tratamento simbólico requer técnicas muito mais especializadas. Os polinómios, com a excepção de alguns casos especiais, não podem ser decompostos em estruturas triangulares básicas. Além do mais a factorização de polinómios é ainda conjectural para polinómios de grau superior a 4 [MOS71]. Os casos especiais possuem assinaturas que estão listadas na Secção 2.4.

Dito isto, consideramos que

> **os polinómios não podem ser resolvidos, simbolicamente, por reescrita axiomática, ou seja, a partir de estruturas triangulares.**

De facto, nesta tese, só foi possível resolver polinómios de 2º grau, analisando a sua estrutura, por completo, e aplicando a fórmula resolvente.

O método de resolução que é apresentado nesta tese, utiliza uma base axiomática totalmente desenvolvida com base numa lógica de 1ª ordem e organizada hierarquicamente por classe e poder redutor. Métodos semelhantes têm sido desenvolvidos mas, na sua grande maioria, baseados em lógicas de ordem superior [BOR02].

## 4.3 A Reescrita de Expressões

A composição lógica de expressões matemáticas, ou composição lógica, por construção-$\lambda$, como lhe chamamos, é apenas um processo de formação de expressões, utilizado tanto na escrita como na reescrita de expressões. A resolução simbólica de expressões envolve a reescrita sistemática de expressões, por via axiomática, ou seja, por aquilo que designamos por reescrita axiomática. A este processo contínuo de transformações lógicas demos o nome de resolução simbólica de expressões matemáticas, por reescrita axiomática.

Como já referimos na Secção 2.5, a reescrita axiomática não é mais do que a substituição de expressões ou subexpressões, por instâncias do lado direito de fórmulas matemáticas, com o lado esquerdo das quais, essas expressões ou subexpressões unificam [BAA98, DER90, DER01]. O conceito de unificação de termos foi abordado na Secção 3.3.2.

Consideramos que

> **as fórmulas matemáticas definem relações de equivalência entre classes de expressões.**

De facto, entre as assinaturas dos lados esquerdos e direitos dessas fórmulas existe uma relação de equivalência. Por exemplo, a assinatura *log_pwr* e *prod_log* são equivalentes.

As relações de equivalência são fechadas para todas as substituições $\sigma$ que resultem da unificação de expressões com os lados esquerdos dessas fórmulas. Segundo Baader e Nipkow [BAA98], se $\rightarrow$ for uma relação de equivalência sobre termos $\mathcal{T}(\Omega,\mathcal{V})$ (Secção 3.3), então diz-se que a relação é fechada, para toda e qualquer substituição $\sigma$, se e só se, para todos os termos $s,t \in \mathcal{T}(\Omega,\mathcal{V})$ e substituições $\sigma$, tem-se que $s \rightarrow t$ implica $\sigma(s) \rightarrow \sigma(t)$. Ou seja, dada uma

fórmula matemática, $l \to r \in \mathcal{B}$, orientada nesse sentido, qualquer instância do seu lado esquerdo, $l\sigma$, é equivalente a uma instância do seu lado direito, $r\sigma$, sob a mesma substituição $\sigma$. A notação $l\sigma$ denota a aplicação $\sigma(l)$, ou seja, a aplicação $\sigma : \mathcal{T}(\Omega, \mathcal{V}) \to \mathcal{T}(\Omega)$, com $l \in \mathcal{T}(\Omega, \mathcal{V})$. Isso pressupõe, naturalmente, que $l \notin \mathcal{V}$ e $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, com $l, r \in \mathcal{T}(\Omega, \mathcal{V})$. Todas as fórmulas matemáticas, incluídas nesta tese, satisfazem a esse requisito.

**Exemplo 4.3-1** ***Fórmula*** $\log(x, \text{pwr}(y, z)) \to \text{prod}(y, \log(x, z))$

*A fórmula* $\log(x, \text{pwr}(y, z)) \to \text{prod}(y, \log(x, z))$, *satisfaz a esse requisito, ou seja,* $\mathcal{V}ar(l) = \{x, y, z\}$ *e* $\mathcal{V}ar(r) = \{x, y, z\}$.

---

É preciso notar que o lado direito das fórmulas é formado pela composição lógica de um certo número de construtores-$\lambda$ e instanciado pela mesma substituição que unificou a expressão e o seu lado esquerdo. Por exemplo, o lado direito da fórmula $\log(x, \text{pwr}(y, z)) \to \text{prod}(y, \log(x, z))$, é instanciado por $\left[\text{prod}(y, u) \circ \log(x, z)\right]_u \sigma$, ou seja, por $\left[\lambda(r)\right]\sigma$, onde $\circ$ representa o operador composição e $u$, a variável de ligação. Nesta tese, a composição lógica, por construção-$\lambda$, é representada, de forma abstracta, por $\left[\lambda(r)\right]$.

### *4.3.1 As Regras de Reescrita*

A base axiomática é um conjunto limitado de regras de reescrita, condicionais (axiomas e teoremas), organizado por classes e com as regras dispostas, hierarquicamente, por classe e por ordem decrescente do seu poder redutor. As regras são seleccionadas, de forma sequencial, com base na assinatura. A assinatura de uma regra é determinada pela estrutura triangular do seu lado esquerdo. Note que uma regra com a assinatura *log* é acedida antes de qualquer

outra com esse mesmo operador, como principal, na sua assinatura.

**Exemplo 4.3-2** ***Regra de reescrita*** $x+0 \rightarrow x$

*No caso da regra* $x+0 \rightarrow x$, *o lado esquerdo,* $x+0$, *possui uma estrutura do tipo* $\Xi$-sum, *ou seja, uma estrutura com apenas uma componente e cuja assinatura é* sum. *Uma expressão, como* $x+0$ *pode ser, portanto, reduzida, por via axiomática, aplicando-lhe uma regra do tipo* $x+0 \rightarrow x$ *(axioma da neutralidade para a soma). Na base axiomática, essa regra está definida por*

$$\text{sum}(s,t) \rightarrow s \quad \textit{if } s \in \mathcal{T} \setminus \mathbb{N} \wedge t \in \{0\}$$

*O resultado da aplicação é uma estrutura do tipo* $\Xi$-none, *ou seja, uma expressão irredutível. Uma transformação lógica deste tipo possui um elevado poder redutor.*

---

Como se pode ver na Figura 4.3.1, a aplicação dessa regra reduz, de facto, o grau de complexidade dessa expressão. Regras desse tipo foram implementadas, em Prolog, por um conjunto de cláusulas do tipo ***apply_relation*** / 7.


**Figura 4.3.1** ***Aplicação do axioma da neutralidade da soma***

Mas nem todas as transformações lógicas, como é óbvio, são assim tão redutoras como essa. De facto, muitas delas são apenas potencialmente redutoras. O que se pode dizer é que todas elas contribuem para a resolução simbólica da expressão através da redução *redexes* ou a formação de novos *redexes*. Por exemplo,

a transformação lógica $\log_e 4^2 \rightarrow \log_e (2^2)^2$ é potencialmente redutora. De facto, o lado direito dessa transformação é potenciadora da formação de um novo *redex*. Essa expressão possui uma estrutura do tipo $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr\_pwr][\Xi\text{-}pwr]$, com um maior número de componentes, mas que é também redutível. A factorização da base da potência introduziu uma nova estrutura de classe à sua classe de reescrita, mas introduziu também a possibilidade de se formarem novos *redexes*. A ideia central por detrás da resolução simbólica é precisamente esta dualidade entre consumidores e produtores de estruturas, ou seja, entre a redução e a formação de *redexes*. A formação de *redexes* é controlada pelo princípio de exclusão e, portanto, restrita à formação de novos *redexes* apenas.

Como vimos atrás, na Secção 2.4.2, quanto maior for o número de componentes a resolver, tanto maior será o grau de complexidade de uma expressão. Pois, com o aumento do número de componentes, um maior número de assinaturas é gerada por reescrita, e maior é, portanto, a sua classe de reescrita.

Mas como veremos, a maioria das transformações lógicas deste tipo, com a excepção das regras de cálculo como, por exemplo, o cálculo de derivadas, são, reescritas contextualizadas, isto é, reescritas onde a regra é apenas aplicada no contexto de um outro operador. Numa reescrita contextualizada, o argumento é transformado como parte integral do contexto. Por exemplo, a conversão de uma raiz só faz sentido como parte de um contexto como o logaritmo ou de uma outra raiz. Como é óbvio, para que uma transformação lógica tenha lugar é necessário que, na base axiomática, existam regras com essa assinatura e que da sua aplicação não resultem expressões ou estados já visitados. Para garantir isso e evitar, portanto, problemas de terminação, é necessário manter em memória o traço da resolução, ou seja, os estados por onde a resolução já passou. Como veremos, o traço de uma resolução não é mais do que a sua memória colectiva.

Um outro aspecto importante a considerar neste tipo de resoluções é

também o poder redutor de uma regra (Secção 2.4.2), ou seja, a sua capacidade de reduzir, ou de contribuir para a redução do grau de complexidade de expressões.

Para garantir que uma regra, $l \to r$, seja redutora ou potencialmente redutora, é necessário que a regra satisfaça aos seguintes critérios :

***Redutora:***

- $\mathcal{N}(l) > \mathcal{N}(r)$, isto é, o lado direito da regra possuir uma estrutura triangular de nível inferior. Por exemplo, no caso da regra $u + 0 \to u$, o lado direito possui um nível, $\mathcal{N}(\Xi\text{-}none)$, inferior ao do seu lado esquerdo, $\mathcal{N}(\Xi\text{-}prod)$;
- $\mathcal{N}(l) = \mathcal{N}(r)$ e $\mathcal{NV}arX(l) > \mathcal{NV}arX(r)$, isto é, o lado direito é do mesmo nível, mas com um número inferior de ocorrências da variável de substituição, $\mathcal{NV}arX$. Por exemplo, no caso das regras, $u + u \to 2u$ e $u(u) \to u^2$, os lados são do mesmo nível, $\mathcal{N}(\Xi\text{-}sum) = \mathcal{N}(\Xi\text{-}prod)$ e $\mathcal{N}(\Xi\text{-}prod) = \mathcal{N}(\Xi\text{-}pwr)$, mas onde $\mathcal{NV}arX(l) = 2$ e $\mathcal{NV}arX(r) = 1$;
- $\mathcal{N}(l) = \mathcal{N}(r)$ e $\mathcal{O}perator(r) = prod$, isto é, o lado direito é do mesmo nível, mas o operador principal é do tipo $prod$. Por exemplo, no caso da regra $\log_a u^k \to k \log_a u$, ambos os lados são do mesmo nível, $\mathcal{N}(\Xi\text{-}log\_pwr) = \mathcal{N}(\Xi\text{-}prod\_log)$, mas o operador principal do seu lado direito é do tipo $prod$;
- $\mathcal{N}(r) > \mathcal{N}(l)$, $\mathcal{O}perator(r) = prod$ e $\mathcal{F}actor(r,1) = const$, isto é, o lado direito é de nível superior, mas o seu operador principal do tipo $prod$ e o seu 1º factor, uma constante simbólica. Por exemplo, no caso da regra $\log_a \sqrt[k]{u} \to \frac{1}{k} \log_a u$, o lado direito é de nível superior, $\mathcal{N}(\Xi\text{-}prod\_div\_log) > \mathcal{N}(\Xi\text{-}log\_root)$, mas o seu operador principal é

do tipo *prod* e $\frac{1}{k}$ é uma constante.

***Potencialmente Redutora:***

- $\mathcal{N}(r) > \mathcal{N}(l)$ e $\mathcal{NR}edex(r) > \mathcal{NR}edex(l)$, isto é, o lado direito é de nível superior, mas o seu lado direito possui mais *redexes*. Por exemplo, no caso da regra $\frac{\mathrm{d}}{\mathrm{dx}}(u+v) \to \frac{\mathrm{d}}{\mathrm{dx}}u + \frac{\mathrm{d}}{\mathrm{dx}}v$, o lado direito é de nível superior, $\mathcal{N}(\Xi\text{-}sum\_der\_der) > \mathcal{N}(\Xi\text{-}der\_sum)$, mas possui mais *redexes*.

Note que, numa base axiomática, não podem existir regras, cujo lado direito é de nível superior, $\mathcal{N}(r) > \mathcal{N}(l)$, mas com o mesmo número ou um número inferior de *redexes*, $\mathcal{NR}edex(r) \leq \mathcal{NR}edex(l)$. Pois estaríamos, simplesmente, a aumentar o grau de complexidade de uma expressão e a diminuir o seu poder redutor. O que seria uma violação do princípio "*reduzir ou aumentar para reduzir*", ou seja, permitir que "*o grau de complexidade de uma expressão possa ser agravado sem uma forte contrapartida de redução*".

Consideramos, portanto, que

> **a aplicação de uma regra de reescrita deve apenas eliminar *redexes* ou potenciar a formação de novos *redexes*.**

Por exemplo, só se deve aplicar a regra $u(v+w) \to u(v)+u(w)$ (distributividade à esquerda em relação à soma), se $u(v)$ ou $u(w)$ forem *redexes* e $v+w$ não o for. Para se garantir isso, tem-se que verificar que não existem redexes por resolver e que existem afinidades operacionais com pelo menos um dos seus argumentos. A aplicação desta propriedade é um exemplo de como o aumento do grau de complexidade de uma expressão deve ser compensado por um aumento do

seu poder redutor.

**Exemplo 4.3-3** ***Resolução simbólica de*** $x(x+1)-x^2$

*Uma expressão, como* $x(x+1)-x^2$*, cuja estrutura é do tipo* $[\Xi\text{-}diff\_prod\_pwr][\Xi\text{-}prod\_sum][\Xi\text{-}pwr]$*, não possui nenhum redex. Para a simplificar é necessário formar um ou mais redexes, ou seja, determinar se as estruturas* $\Xi\text{-}prod\_sum$ *e* $\Xi\text{-}pwr$ *possuem alguma afinidade operacional.*

*Neste caso, a aplicação da regra* $u(v+w)\rightarrow u(v)+u(w)$ *à sua componente* $x(x+1)$*, daria origem a uma expressão equivalente,* $(x^2+1)-x^2$*, cuja estrutura é do tipo* $[\Xi\text{-}diff\_sum\_pwr][\Xi\text{-}sum\_pwr][\Xi\text{-}pwr]^2$*, mais complexa, mas onde é possível associar as estruturas* $\Xi\text{-}pwr$ *e formar um redex. A aplicação de uma regra do tipo* $(u+v)-w\rightarrow(u-w)+v$ *(associatividade e comutatividade) daria origem à formação do redex* $x^2-x^2$*.*

---

Mas a aplicação de regras de reescrita (axiomas e teoremas) é o resultado da aplicação daquilo que poderíamos designar por estratégias de resolução. A resolução simbólica de expressões envolve não só a selecção de regras de rescrita a aplicar como também a selecção das posições onde aplicar essas regras. Numa estrutura triangular, essas posições coincidem, precisamente, com as posições ocupadas por cada uma das suas componentes, ou seja, com o topo das suas respectivas estruturas triangulares.

O topo de uma estrutura triangular $\Xi$ é representado pelo seu operador

principal. Uma estrutura triangular é uma estrutura binária, $op(arg, arg)$, com três (3) vértices.

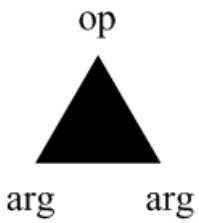



**Figura 4.3.2** *Operador principal*

O operador $op$ é o topo dessa estrutura e os seus dois argumentos $arg$, a sua base. Por convenção, uma estrutura triangular é analisada, recursivamente, a partir do topo, prosseguindo, em profundidade, da esquerda para a direita. Por exemplo, para se resolver uma expressão, como $\log(e, \text{pwr}(2,4))$, começa-se primeiro por analisar a estrutura $\log(arg,arg)$, e só depois, de forma recursiva, a estrutura $\text{pwr}(arg, arg)$.

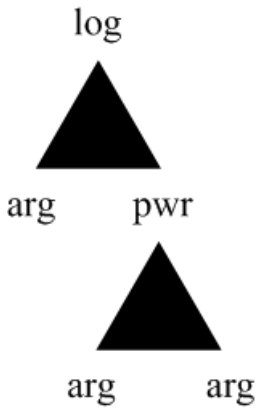



**Figura 4.3.3** *Operador argumento*

As estratégias, apresentadas nesta tese, visam essencialmente a redução do grau de complexidade das expressões. As reduções são transformações lógicas, $\Gamma_R$, efectuadas, por reescrita axiomática, que visam, essencialmente, a

normalização dessas expressões. As reduções são aplicadas, de preferência, aos *redexes* mais interiores. Por exemplo, na expressão $(3+4)+0$ , o *redex* $3+4$ seria o primeiro a ser resolvido. A solução, neste caso, seria

$$(3+4)+0 \rightarrow 7+0 \rightarrow 7$$

Cada classe possui a sua própria base axiomática . A base axiomática para estruturas $\Xi$-*log* , $\mathcal{B}_{log}$ , é um subconjunto da base $\mathcal{B}_{log\_pwr}$ , para estruturas $\Xi$-*log_pwr* , ou seja, $\mathcal{B}_{log} \subset \mathcal{B}_{log\_pwr}$ . Assim, para resolver uma expressão, como $\log(e, \text{pwr}(2,4))$ , é necessário aplicar duas bases axiomáticas; uma, para logaritmos de potências, $\mathcal{B}_{log\_pwr}$ e outra para potências, $\mathcal{B}_{pwr}$ .

Como veremos, as estratégias de resolução estão organizadas por operador.

Consideramos que

> **a resolução simbólica de expressões não é mais do que uma aplicação estratégica de uma ou mais classes de reescrita.**

De facto, a classe de reescrita de uma expressão é a união das classes de reescrita de todas as suas componentes. A resolução simbólica de uma expressão é portanto o resultado da resolução de todas essas componentes e da sua composição de acordo com a estratégia aplicada.

Consideramos também que

> **a resolução simbólica de expressões não depende da ordem em que as componentes são resolvidas, ou seja, da ordem em que as regras são aplicadas.**

De facto, a ordem em que as regras são aplicadas afecta apenas a solução e não o resultado. Se a base for completa, o resultado da resolução é a forma mais simples da expressão.

**Exemplo 4.3-4** ***Resolução simbólica de*** $((1+2)+3)+4$

*No caso da expressão* $((1+2)+3)+4$, *cuja estrutura é do tipo* $[\Xi\text{-}sum\_sum]^2[\Xi\text{-}sum]$, *a estratégia sugerida, como preferencial, é a redução da componente* $[\Xi\text{-}sum]$, *ou seja, a execução de um cálculo numérico. De facto, essa seria a transformação lógica com maior poder redutor. O único redex presente nessa expressão é, na realidade, a expressão* $1+2$.

*Como resultado, a expressão é simplificada para* $(3+3)+4$, *cuja estrutura é do tipo* $[\Xi\text{-}sum\_sum][\Xi\text{-}sum]$, *mais simples. De facto, com essa transformação, um dos operadores soma,* $\Xi\text{-}sum$, *foi eliminado.*

---

Como veremos na Secção 4.3.1, as regras de reescrita, por cálculo, são regras com um elevado poder redutor. Como estratégia, a presença de *redexes* sugere sempre, como prioritária, a aplicação de estratégias estritas. Mas, como veremos também, nem sempre estão presentes *redexes*, nessas expressões. Daí a importância, numa reescrita axiomática, de se poderem formar *redexes*, que possam contribuir para a resolução simbólica dessas expressões. Mas para se formar *redexes*, é necessário aumentar o grau de complexidade e o poder redutor dessas expressões, ou seja, aplicar regras que designamos por potencialmente redutoras. Essas compensam o aumento do grau de complexidade com a formação de um ou mais *redexes*. É importante, que essas regras formem mais *redexes* do que já existem. Regras, com essas características, potenciam reduções futuras.

**Exemplo 4.3-5** ***Resolução simbólica de*** $(x+1)+(x+2)$

*Para simplificar a expressão* $(x+1)+(x+2)$, *é necessário formar redexes, ou*

*seja, associar os termos semelhantes [BUN85], aplicando regras de conversão (Secção 4.3.1.4) do tipo*

**R1.** $\text{sum}(\text{sum}(s,t),\text{sum}(q,r)) \rightarrow \text{sum}(\text{sum}(s,q),\text{sum}(t,r))$

***if*** $s,q \notin \mathbb{N} \wedge t,r \notin \{0\} \wedge akin(s,q)$

**R2.** $\text{sum}(\text{sum}(s,t),\text{sum}(q,r)) \rightarrow \text{sum}(\text{sum}(s,q),\text{sum}(t,r))$

***if*** $s,q \notin \mathbb{N} \wedge t,r \in \mathbb{N} \bigcup \mathcal{K} \setminus \{0\} \wedge akin(t,r)$

*A aplicação da regra **R**2 conduz a uma expressão equivalente,* $(x+x)+(1+2)$*, com o mesmo grau de complexidade, mas onde estão presentes dois redexes. Essa expressão é reduzida à sua formal normal,* $2x+3$*, que neste caso é também a mais simples.*

---

O número de regras deste tipo (neste caso, associatividade) é bastante reduzido, pois apenas são considerados os casos onde existe a possibilidade de se formarem *redexes*.

Numa resolução simbólica, a estratégia preferencial é a redução dos *redexes*, começando pelos mais interiores. Nesta tese, optou-se por uma busca em profundidade, da esquerda para a direita.

**Exemplo 4.3-6** ***Resolução simbólica de*** $\frac{\mathrm{d}}{\mathrm{dx}}(x+4)$

*No caso da expressão* $\frac{\mathrm{d}}{\mathrm{dx}}(x+4)$*, cuja estrutura é do tipo* $[\Xi\text{-}der\_sum][\Xi\text{-}sum]$*, não existem redexes. A estratégia sugerida, como*

*preferencial, é a transformação da sua componente principal* $\Xi\text{-}der\_sum$, *ou seja, a aplicação de uma regra de derivação com essa assinatura. O resultado seria a seguinte cadeia de transformações*

$$\frac{\mathrm{d}}{\mathrm{dx}}(x+4) \rightarrow_{\Gamma_1} \frac{\mathrm{d}}{\mathrm{dx}}x + \frac{\mathrm{d}}{\mathrm{dx}}4 \rightarrow_{\Gamma_2} 1 + \frac{\mathrm{d}}{\mathrm{dx}}4 \rightarrow_{\Gamma_3} 1+0 \rightarrow_{\Gamma_4} 1$$

*Como se pode observar, o resultado da aplicação,* $\Gamma_1$, *é uma expressão com uma estrutura do tipo* $[\Xi\text{-}sum\_der\_der][\Xi\text{-}der]^2$, *mais complexa, mas onde estão presentes dois redexes.*

---

**Exemplo 4.3-7** ***Resolução simbólica de*** $\frac{\mathrm{d}}{\mathrm{dx}}(x+x)$

*No caso da expressão* $\frac{\mathrm{d}}{\mathrm{dx}}(x+x)$, *com o mesmo tipo de estrutura, mas com um redex presente, a estratégia sugerida, como preferencial, seria a redução da estrutura* $\Xi\text{-}sum$. *O resultado, neste caso, seria a seguinte cadeia de transformações*

$$\frac{\mathrm{d}}{\mathrm{dx}}(x+x) \rightarrow_{\Gamma_1} \frac{\mathrm{d}}{\mathrm{dx}}(2x) \rightarrow_{\Gamma_2} 2\frac{\mathrm{d}}{\mathrm{dx}}x \rightarrow_{\Gamma_3} 2(1) \rightarrow_{\Gamma_4} 2$$

---

**Exemplo 4.3-8** ***Resolução simbólica de*** $\frac{\mathrm{d}}{\mathrm{dx}}(\log_e x^2 - 2\log_e x)$

*No caso da expressão* $\frac{\mathrm{d}}{\mathrm{dx}}(\log_e x^2 - 2\log_e x)$, *cuja estrutura é do tipo*

$$[\Xi\text{-}der\_diff\,][\Xi\text{-}diff\_log\_prod][\Xi\text{-}log\_pwr][\Xi\text{-}prod\_log][\Xi\text{-}pwr][\Xi\text{-}log],$$

*o redex mais interior é a expressão com a estrutura* $[\Xi\text{-}log\_pwr]$. *Esta é, portanto, a primeira componente a ser resolvida. Com esta resolvida, a expressão passa a ter uma estrutura do tipo*

$$[\Xi\text{-}der\_diff\,][\Xi\text{-}diff\_prod\_prod][\Xi\text{-}prod\_log]^2[\Xi\text{-}log]^2$$

*O redex mais interior, neste caso, é a expressão com a estrutura* $[\Xi\text{-}diff\_prod\_prod]$. *O resultado da resolução é a seguinte cadeia de transformações*

$$\frac{\mathrm{d}}{\mathrm{dx}}(\log_e x^2 - 2\log_e x) \rightarrow_{\Gamma_1} \frac{\mathrm{d}}{\mathrm{dx}}(2\log_e x - 2\log_e x) \rightarrow_{\Gamma_2} \frac{\mathrm{d}}{\mathrm{dx}}0 \rightarrow_{\Gamma_2} 0$$

---

Verificámos que em todos os casos de resolução, estudados para esta tese, o resultado foi sempre a forma mais simples dessas expressões. Não podemos garantir que a base axiomática, utilizada nesta tese, esteja completa. Podemos, no entanto, garantir que é possível completar essa base a partir dos axiomas e teoremas listados no Apêndice B e que, portanto, o processo de resolução, apresentado nesta tese, é convergente. A unicidade da forma normal depende apenas da completude da base axiomática e é garantida pelo princípio de exclusão através da memória colectiva.

Consideramos, portanto, que

> **a resolução simbólica de expressões é um processo contínuo de reescritas axiomáticas, envolvendo reduções, conversões e simplificações, onde as estruturas triangulares são consumidas e alimentadas.**

De facto, a reescrita axiomática não é a mais do que um processo de consumo e produção de redexes, controlado pelo princípio de exclusão e por *backtracking*, que visa, essencialmente, a reescrita de formas equivalentes que possam contribuir para a resolução simbólica de expressões.

**Definição 4.3-1** ***Reescrita Condicional***

> *Uma reescrita condicional é a aplicação de uma regra de reescrita condicional, do tipo* $l \to r$ **if** C*, cujo resultado é a transformação lógica* $s \to t$*, isto é, a substituição de uma expressão* $s$*, unificável (Secção 3.3.2) com o lado esquerdo* $l$ *da regra, por uma instância* $t \equiv r\sigma$ *do seu lado direito* $r$*. A substituição* $\sigma = \text{mgu}(s,l)$ *é o unificador mais geral (Secção 3.3.2).*

A aplicação de fórmulas matemáticas (axiomas e teoremas) são formas de reescrita condicional. A transformação lógica $\log_e 2^4 \to 4\log_e 2$ é o resultado de uma reescrita condicional, não contextualizada. Como é óbvio, uma reescrita pode ser efectuada em qualquer posição da estrutura, desde que essa parte da estrutura possua a assinatura adequada para essa reescrita.

**Definição 4.3-2** ***Reescrita Contextualizada***

> *Diz-se que uma reescrita é contextualizada se a aplicação da regra depende do contexto em que a reescrita é efectuada. O contexto é definido pelo operador principal da expressão cujo argumento é o objecto da reescrita .*

A factorização de números inteiros positivos é uma forma de reescrita contextualizada. A transformação lógica $\log_e 16 \to \log_e 2^4$, é o resultado de uma reescrita contextualizada pelo termo funcional $\log(\_,\_)$.

Nesta tese, foram identificados os seguintes tipos de reescrita: redução,

conversão, simplificação e composição lógica.

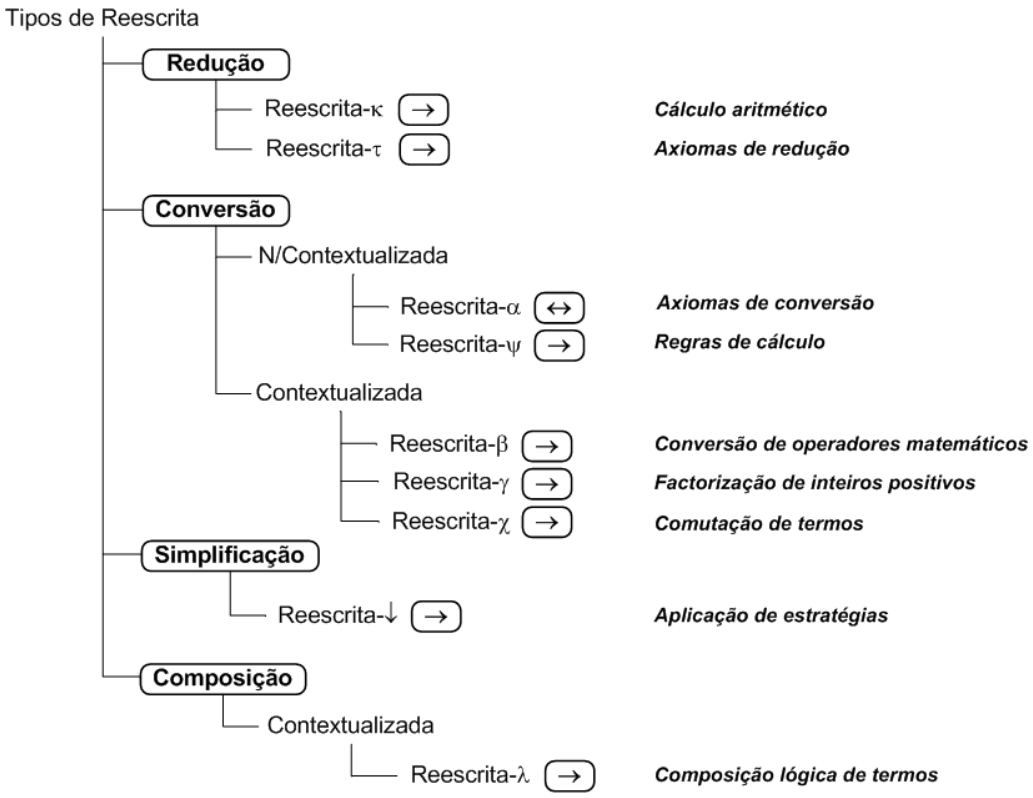



**Figura 4.3.4** *Tipos de reescrita*

## *Redução:*

- **Reescrita por cálculo numérico**, ou **reescrita-$\kappa$** (Secção 4.3.1.1) – A aplicação de cálculos numéricos[39]. Numa reescrita-$\kappa$, a expressão numérica é substituída pelo seu valor numérico ou simbólico. O resultado de um cálculo numérico é uma expressão irredutível. Por exemplo, $2+3 \to 5$ ou $-4+2 \to -2$ são reescritas deste tipo. A reescrita por cálculo possui um elevado poder redutor;
- **Reescrita por redução axiomática**, ou **reescrita-$\tau$** (Secção

[39] Os cálculos numéricos são, essencialmente, reduções por cálculo.

4.3.1.2) – A aplicação directa de reduções. Tal como o cálculo numérico, a reescrita-$\tau$ possui um elevado poder redutor. O resultado de uma redução é uma expressão irredutível. Por exemplo, $x+x \to 2x$ é uma reescrita deste tipo.

### *Simplificação:*

– **Reescrita por simplificação**, ou **reescrita**-$\downarrow$ (Secção 4.3.1.3) – A aplicação indirecta de reduções ou conversões. A reescrita-$\downarrow$ é aquilo que poderíamos designar por reescrita indirecta. Por exemplo, $x+(2+3) \to x+5$ é uma reescrita deste tipo. A reescrita é aplicada a uma das componentes da estrutura e a estrutura completa reescrita por composição lógica de construtores-$\lambda$.

### *Conversão:*

– **Reescrita por conversão axiomática**, ou **reescrita**-$\alpha$, -$\beta$, -$\psi$, -$\gamma$ e -$\chi$ (Secção 4.3.1.4) – A aplicação directa de fórmulas matemáticas à posição $p \in \mathcal{P}os$ de uma estrutura, onde $\mathcal{P}os = \{i \mid i \in \mathbb{N}\}$.

- **Conversão unidireccional não contextualizada**, ou **reescrita** -$\psi$ - A aplicação unidireccional de fórmulas matemáticas. A aplicação de uma regra de derivação ou a decomposição de uma derivada de ordem superior são reescritas deste tipo. Por exemplo, a transformação $\frac{\mathrm{d}}{\mathrm{dx}}(x+4) \to \frac{\mathrm{d}}{\mathrm{dx}}x + \frac{\mathrm{d}}{\mathrm{dx}}4$ é uma reescrita-$\psi$;
- **Conversão unidireccional contextualizada**, ou **reescrita**-$\gamma$ - A factorização de números inteiros positivos, devidamente

contextualizados. Este tipo de reescrita é potencialmente redutor. Por exemplo, a transformação $\log_e 16 \rightarrow \log_e 2^4$ é uma reescrita $-\gamma$. O contexto é definido pela estrutura $\Xi$-*log* .

- **Conversão bidireccional não contextualizada**, ou **reescrita** $-\alpha$ - A aplicação de fórmulas em ambos os sentidos. Um dos sentidos possui maior poder redutor. A aplicação de relações fundamentais ou a aplicação das propriedades associativa ou distributiva são reescritas deste tipo. Por exemplo, a transformação $\log_e x^2 \rightarrow 2\log_e x$ é uma reescrita $-\alpha$. A aplicação no sentido oposto está sujeita a condições especiais, como por exemplo, a existência de afinidades operacionais entre os seus argumentos;
- **Conversão bidireccional contextualizada**, ou **reescrita** $-\beta$ - A aplicação bidireccional de fórmulas matemáticas a estruturas devidamente contextualizadas. Este tipo de reescrita é potencialmente redutora. A conversão de operadores é um exemplo deste tipo de reescrita. Por exemplo, a transformação $\log_e \sqrt{x} \rightarrow \log_e x^{\frac{1}{2}}$ é uma reescrita $-\beta$. O contexto é definido pela estrutura $\Xi$-*log_root* .
- **Conversão unidireccional, por comutação**, ou **reescrita** $-\chi$ – A aplicação da propriedade comutativa. Tal como a reescrita $-\gamma$ e $-\lambda$, este tipo de rescrita é também contextualizado. Por exemplo, a transformação $2 + x \rightarrow x + 2$ é um tipo de reescrita $-\chi$.

***Composição:***

- **Reescrita por composição lógica**, ou **reescrita-$\lambda$** – A formação de estruturas triangulares por conjunção lógica de construtores-$\lambda$. Tal como a reescrita $\gamma$, este tipo de reescrita é também contextualizado. Por exemplo, a transformação $\left[\log_e X\right]_{X=4^2} \rightarrow \log_e 4^2$ é um tipo de reescrita -$\lambda$. O contexto é definido pela regra onde a composição tem lugar;

Nesta tese, as regras de reescrita são identificadas pelo tipo de reescrita que efectuam (*prefixo*) e pela classe de expressões que reescrevem (*assinatura*). No caso de regras, com estruturas de nível 1, o sufixo *arg* ou *args* é adicionado ao nome da regra para identificar se apenas um, ou ambos, os argumentos estão envolvidos. A regra $x+0 \rightarrow x$, por exemplo, é identificada pelo nome *relate_sum_arg*, pois está relacionada com uma característica de um dos seus argumentos ser $0$ (seja ele qual for). A regra $x+x \rightarrow 2x$, por outro lado, é identificada pelo nome *relate_sum_args*, pois está relacionada com uma característica que diz respeito a ambos dos seus argumentos (serem, neste caso, idênticos).

### *4.3.1.1 A Reescrita por Cálculo Numérico*

Nesta tese, o cálculo numérico é apenas efectuado sobre o conjunto dos números inteiros $\mathcal{I}$. Expressões, como $2^{-2}$ não são redutíveis por cálculo, pois representam números racionais $\mathcal{Q}$. A redução só pode ser feita, por via axiomática, ou seja, reduzindo a expressão à constante simbólica $\frac{1}{4}$. O mesmo se pode dizer de expressões, como $(-2)^2$. Para as reduzir é necessário transformá-las, primeiro, por via axiomática, e só depois aplicar o cálculo numérico. Essa expressão teria de passar pelas seguintes transformações

$$(-2)^2 \rightarrow (-1)^2 2^2 \rightarrow 1(4) \rightarrow 4$$

A expressão $-2$ é interpretada como a simétrica de $2$.

Os números racionais $\mathcal{Q}$ e irracionais $\bar{\mathcal{Q}}$ são tratados como constantes simbólicas, isto é, $\mathcal{Q} \subset \mathcal{K}$ e $\bar{\mathcal{Q}} \subset \mathcal{K}$. Por exemplo, a fracção $\frac{2}{5}$ é tratada como uma constante simbólica do tipo $\Xi$-*div* e o a raiz $\sqrt{3}$, como uma constante simbólica do tipo $\Xi$-*root*.

O cálculo numérico é um tipo de transformação lógica com um elevado poder redutor. O resultado de um cálculo numérico é sempre uma expressão com uma estrutura triangular do tipo $\Xi$-*none* (constante numérica) ou $\Xi$-*prod* (constante simbólica), ou seja, uma expressão na sua forma mais simples. O cálculo numérico é uma forma de transformação que envolve apenas reduções. A este tipo de reescrita demos o nome de reescrita-$\kappa$.

O cálculo numérico é encarado, nesta tese, como uma forma especial de reescrita. O cálculo, propriamente dito, é efectuado pelo predicado Prolog[40], ***is*** / 2, utilizando os operadores aritméticos, $\oplus \in \{+,-,*,**\}$, fornecidos pelo Prolog. A resolução do literal, $u$ **is** $s \oplus t$, é encarada, nesta tese, como uma forma de reescrita de expressões numéricas, ou seja, $s \oplus t \rightarrow u$. Esse predicado é apenas aplicável aos números naturais, $\mathbb{N}$. Cálculos envolvendo números racionais, $\mathcal{Q}$, e irracionais, $\bar{\mathcal{Q}}$, são tratados apenas de forma simbólica. O tratamento estritamente lógico do cálculo aritmético, utilizando a função sucessor, é bastante elegante, mas não muito prático [STE00].

A reescrita-$\kappa$ é apenas aplicável a expressões numéricas com estruturas

[40] Os predicados do sistema ou *bips* ("***b**uilt-in **p**redicate**s**"*) [STE00] são predicados fornecidos pelo Prolog cujo papel é facilitar o acesso às capacidades aritméticas do computador de uma forma mais directa . A alternativa seria o tratamento estritamente lógico e axiomático do cálculo a partir de uma definição lógica dos números naturais.

$\Xi$-*op* , onde $op \in \mathcal{S}ig^{\kappa}$ e

$$\mathcal{S}ig^{\kappa} = \{sum, diff, prod, pwr\}$$

$$\mathcal{C}^{\kappa} = \{ f(s,t) \in \mathcal{T} \mid f \in \mathcal{S}ig^{\kappa} \wedge s,t \in \mathcal{I} \}$$

**Definição 4.3-3** ***Regra de Reescrita*** $\kappa$

> *Uma regra de reescrita* $\kappa$ *é uma regra de reescrita condicional, representada por* $\rightarrow_{\kappa}$, *que efectua transformações lógicas do tipo* $s \rightarrow [\lambda_{\kappa}(r)]\sigma$, *onde* $s \in \mathcal{C}^{\kappa}$ *e* $\lambda_{\kappa}(r)$ *representa uma reescrita* $\kappa$.

As regras de reescrita $\kappa$ são regras de reescrita, condicionais, baseadas no predicado ***evaluate*** / 7 [41]. As regras de reescrita $\kappa$ estão organizadas por classes $c \in \mathcal{C}^{\kappa}$. A aplicação de uma regra de reescrita $\kappa$ pode ser representada simbolicamente por uma expressão do tipo

$$\textbf{evaluate}(L \rightarrow R, State, Rule) \leftarrow$$

$$C_1, \ldots, C_n,$$

$$\lambda_{\kappa}(R).$$

O termo $L$ é o termo livre que representa a estrutura lógica do lado esquerdo da fórmula a aplicar (cálculo aritmético, neste caso). O termo $R$ é a variável lógica que representa o resultado de uma reescrita $\kappa$, ou seja, a expressão $[\lambda_{\kappa}(R)]\sigma$. O termo $C_i$ é o literal que representa a $i$-gésima condição de aplicabilidade dessa regra. O termo $\lambda_{\kappa}(R)$ é o literal que representa a reescrita $\kappa$.

[41] O termo $L \rightarrow R$ é apenas uma forma mais abstracta de representar uma regra de reescrita. Em Prolog, esse termo é representado por um termo livre, $L$, e uma variável lógica, $R$. Para tornar a leitura mais apelativa, optámos também por utilizar o símbolo $\leftarrow$, em vez de :- , como forma de representar uma implicação ao inverso.

O termo *State* é o termo livre que identifica a parte da estrutura que se pretende reescrever. O termo *Rule* é o termo livre que identifica a fórmula aplicada.

Uma reescrita-$\kappa$, denotada por $\lambda_{\kappa}(R)$, é uma transformação lógica, por cálculo, que resulta da resolução de um literal do tipo ***is*** / 2, ou seja, de um literal do tipo $u$ **is** $s \oplus t$, onde $\otimes$ representa um operador aritmético Prolog, ou da conjunção de um literal desse literal e um construtor $\lambda$-*prod*. A expressão $1+2$ é resolvida por um literal desse tipo, ou seja,

$$R \textbf{ is } S+T, \textit{ com } S=1 \textit{ e } T=2$$

Uma reescrita-$\kappa$ reduz expressões numéricas $s \in \mathcal{C}^{\kappa}$.

**Exemplo 4.3-9** ***Reescrita*** $\kappa$-*sum* ***(Somas positivas)***

*No caso da expressão* $1+2$, *a reescrita é designada por* $\kappa$-*sum*. *Uma reescrita* $\kappa$-*sum é uma transformação lógica do tipo*

$$1+2 \rightarrow_{\kappa} 3$$

*A regra* $\kappa$-*sum aplicada, neste caso, é uma regra de reescrita do tipo*

**R1.** $\text{sum}(s,t) \rightarrow r$ ***if*** $s,t \in \mathbb{N} \wedge r = s+t$

---

A implementação Prolog de uma regra $\kappa$-***sum*** (Programa A.1-1) está listada na Secção A.1.1 do Apêndice A.

No caso de somas numéricas, cujos resultados são negativos, a reescrita $\kappa$-*sum* utiliza a composição lógica, por construção-$\lambda$ (Secção 4.2.2), para reescrever inteiros negativos.

**Exemplo 4.3-10** ***Reescrita*** $\kappa$***-sum*** ***(Somas negativas)***

*No caso da reescrita* $\kappa$*-sum* $-4+2 \to -2$*, o resultado* $-2$ *é representado pelo termo* $\text{prod}(-1,2)$*. A reescrita desse termo envolve um construtor* $\lambda$*-prod .*

*A regra* $\kappa$*-sum aplicada, neste caso, é uma regra de reescrita do tipo*

**R1.** $\text{sum}(s,t) \to \text{prod}(-1,r)$ ***if*** $s \in \mathcal{I}^- \wedge t \in \mathbb{N} \wedge |s| > t \wedge r = |s| - t$

---

As regras de reescrita-$\kappa$ são regras com um elevado poder redutor (Secção 2.4.2). De facto, essas regras reduzem expressões numéricas, envolvendo números inteiros, a constantes numéricas ou simbólicas.

### *4.3.1.2 A Reescrita por Redução Axiomática*

As reduções, por via axiomática, são transformações lógicas baseadas na aplicação de fórmulas matemáticas com elevado poder redutor. Essas fórmulas são apenas aplicadas num único sentido. Fórmulas que designamos por unidireccionais. Um exemplo típico é a fórmula que descreve o axioma da neutralidade da soma. Esse axioma pode ser representado, simbolicamente, por uma expressão do tipo

$$x + 0 = x$$

Neste tipo de expressões, a variável matemática $x$ [42] representa apenas uma variável lógica $x \in \mathcal{V}$ . As fórmulas matemáticas estão representadas, nesta tese, por um par determos livres, $l, r \in \mathcal{T}(\Omega, \mathcal{V})$ .

[42] Sempre que possa haver ambiguidade entre esses dois conceitos de variáveis, utilizaremos a designação "*variável matemática*" para referir a variáveis de cálculo ou incógnitas $x$ , e a designação "*variável lógica*" para referir a variáveis de substituição $x$ . Quando não qualificada, a designação "*variável*" referir-se-á sempre, à variável matemática.

No caso do axioma da neutralidade para a soma, o símbolo $0$ representa o elemento neutro da adição. Computacionalmente, o que esse axioma diz é que o valor de uma expressão soma não se altera, se o elemento 0 for adicionado, ou removido dessa expressão. Como regra de reescrita, porém, $x+0 \to x$ exprime mais do que isso. A aplicação dessa regra reduz, de facto, o grau de complexidade dessa expressão (Secção 2.4.2). Com a *eliminação* desse elemento, o próprio operador "$+$" é também eliminado. A aplicação dessa regra torna a expressão mais simples (Secção 2.4.2). Já o mesmo não sucede com a aplicação de uma regra, do tipo $x \to x+0$.

A questão aqui não é apenas uma questão de orientação de fórmulas. É também uma questão computacional. Seja qual for a orientação que se dê a uma fórmula, é necessário lidar com o seu aspecto computacional. Pois, nem todas as orientações são computacionalmente viáveis.

Os lados (*esquerdo* e *direito*) de uma fórmula, $l \to r$, são termos livres $l, r \in \mathcal{T}(\Omega, \mathcal{V})$[43]. As variáveis $x_i \in \mathcal{V}ar(l)$ e $x_j \in \mathcal{V}ar(r)$, com $x_i, x_j \in \mathcal{V}$, são variáveis lógicas, ou seja, variáveis que podem intervir em transformações lógicas. As expressões matemáticas $s \in \mathcal{T}$ representam apenas padrões simbólicos com um determinado significado operacional, ou seja, termos constantes $s \in \mathcal{T}(\Omega)$.

O que uma fórmula nos diz, seja ela matemática ou não, é que, entre os seus *lados*, existe, de facto, uma relação de equivalência, e que essa relação pode ser utilizada, em ambos os sentidos, como uma *fórmula* de reescrita de expressões. Se essa fórmula é matemática, então a reescrita é simplesmente uma reescrita de expressões matemáticas. Outros tipos de fórmulas fariam reescritas de outra natureza. Porém, para se utilizar uma fórmula, computacionalmente, é necessário que as expressões, que se pretende reescrever, sejam *unificáveis* (Secção 3.3) com o

[43] Termos com variáveis lógicas.

lado esquerdo da fórmula, ou seja, o lado que serve de antecedente à transformação.

Mas sendo assim, a relação oposta ao do axioma da neutralidade da soma, $x \to x + 0$, representaria também uma transformação lógica possível. Só que neste caso, a transformação teria, computacionalmente, um efeito perverso, pois, aplicando-a, estaríamos a adicionar termos a essa expressão, *ad infinitum*, e, portanto, a criar cadeias de transformação infinitas. Transformações que não são, computacionalmente, viáveis e que Dershowitz et al. [DER01], considera também não terminante.

A aplicação dessas regras teria o mesmo efeito computacional que o produzido pelo axioma da identidade (reflexividade), $x \to x$. Só que, neste caso, a expressão permaneceria indefinidamente a mesma, enquanto que, na outra, cresceria indefinidamente. Tanto uma como a outra, mostram que não se deve utilizar fórmulas cujos antecedentes sejam apenas a variável livre dessas transformações [DER01]. Não se deve, portanto, utilizar regras de reescrita cujos *antecedentes* sejam *unificáveis* (Secção 3.3) com expressões atómicas (variáveis lógicas). Transformações, como $x \to x + 0$, não devem fazer parte da base axiomática de um sistema de reescrita.

Mas, uma transformação lógica, como $x + 0 \to x$, teria, pelo contrário, um efeito oposto, ou seja, um poder redutor bastante elevado. Esse efeito seria, de facto, tão elevado como o de uma redução, por cálculo numérico. A este tipo de reescrita demos o nome de reescrita, por redução axiomática, ou simplesmente reescrita-$\tau$. O efeito de uma reescrita-$\tau$ é sempre terminal (terminante [BAA98, DER01]) . Tanto a reescrita-$\kappa$ como a reescrita-$\tau$ são reduções $\Gamma_R$ (Secção 2.5).

Numa redução, por via axiomática, as fórmulas matemáticas estão orientadas também num único sentido – o sentido redutor. Tal como no caso do axioma da neutralidade, transformações lógicas, como $x + x \to 2x$ e $x(x) \to x^2$,

são consideradas também reduções, mas por via axiomática. E como tal, as transformações opostas não devem participar na reescrita de expressões.

A reescrita-$\tau$ é apenas aplicável a expressões não numéricas com estruturas $\Xi$-$op$, onde $op \in \mathcal{S}ig^{\tau}$ e

$$\mathcal{S}ig^{\tau} = \mathcal{O}p$$

$$\mathcal{C}^{\tau} = \left\{ f(s,t) \in \mathcal{T} \mid f \in \mathcal{S}ig^{\tau} \wedge s,t \in \mathcal{T} \right\}$$

**Definição 4.3-4** ***Regra de Reescrita-$\tau$***

> *Uma regra de reescrita-$\tau$ é uma regra de reescrita condicional, representada por $\rightarrow_{\tau}$, que efectua transformações lógicas do tipo $s \rightarrow [\lambda_{\tau}(r)]\sigma$, onde $s \in \mathcal{C}^{\tau}$ e $\lambda_{\tau}(r)$ representa uma reescrita-$\tau$.*

As regras de reescrita-$\tau$ são regras de reescrita, condicionais, baseadas no predicado ***apply_relation*** / 7. Tal como as outras, também estas estão organizadas por classes $c \in \mathcal{C}^{\tau}$.

A aplicação de uma regra de reescrita$-\tau$ pode ser representada simbolicamente por uma expressão do tipo

**apply_relation**$(L \rightarrow \mathrm{R}, State, Rule) \leftarrow$

$\mathrm{C}_1, \ldots, \mathrm{C}_n,$

$\lambda_{\tau}(R).$

Uma reescrita-$\tau$, denotada por $\lambda_{\tau}(R)$, é uma transformação lógica, por redução axiomática, que resulta da composição lógica de um ou mais literais do tipo ***op*** / 3 (construtores-$\lambda$), onde $op \in \mathcal{O}p$, ou da simples instanciação de uma

variável lógica. O resultado da reescrita $x+0 \rightarrow x$ é a instanciação da variável $s$ do lado esquerdo da fórmula $\text{sum}(s,t)$, por unificação (Secção 3.3.2).

Uma reescrita-$\tau$ reduz expressões da classe $c \in \mathcal{C}^{\tau}$.

**Exemplo 4.3-11** ***Reescrita de*** $\tau$***-sum***

*No caso da expressão* $x+0 \rightarrow_{\tau} x$*, a reescrita é designada por* $\tau$*-sum. Uma reescrita* $\tau$*-sum é uma transformação lógica do tipo*

$$x+0 \rightarrow_{\tau} x$$

*A regra* $\tau$*-sum aplicada, neste caso, é uma regra de reescrita do tipo*

**R1.** $\text{sum}(s,t) \rightarrow s$ ***if*** $s \in \mathcal{T} \setminus \mathcal{I} \wedge t \in \{0\}$

---

A implementação Prolog de uma regra $\tau$***-sum*** está listada na Secção A.1.2 do Apêndice A.

As regras de reescrita-$\tau$ são, também, regras com um elevado poder redutor (Secção 2.4.2). Também elas reduzem o grau de complexidade das expressões.

### *4.3.1.3 A Reescrita por Simplificação*

A reescrita, por simplificação, é um caso especial de transformação. Numa reescrita deste tipo, apenas uma parte da expressão é reescrita, por via axiomática. A outra define apenas o contexto dessa reescrita. A reescrita da expressão, como um todo, é o resultado de uma composição lógica envolvendo a parte reescrita e o respectivo contexto. A reescrita não é mais do que a substituição do argumento do contexto, ou "*hole*", pela parte reescrita. Se $s,t \in \mathcal{T}$ e $t[s]$ for um subtermo

próprio de $t$, então o contexto de $s$ é o termo $t[\bullet]$ onde $\bullet$ é o argumento ou "*hole*".

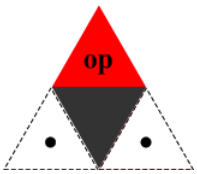



**Figura 4.3.5** ***Contexto***

**Exemplo 4.3-12** ***Reescrita por simplificação***

*A reescrita da expressão* $x+(1+2)$ *é o resultado da seguinte transformação lógica*

$$x+(1+2) \rightarrow x+3$$

*Uma transformação deste tipo envolve mais do que uma reescrita. A expressão* $x+3$ *é, de facto, o resultado de uma redução mas uma que teve lugar, de forma indirecta, por aplicação de uma regra de reescrita-*$\kappa$, $1+2 \rightarrow_{\kappa} 3$.

*A transformação por simplificação só pode ter lugar depois de a expressão* $1+2$ *ter sido reduzida. Como se pode observar, a expressão* $1+2$ *é, de facto, o único redex presente na expressão* $x+(1+2)$.

---

Consideramos que

> **todas as transformações lógicas são consumidoras ou produtoras de *redexes*.**

De facto, o resultado de uma transformação lógica só pode ser uma estrutura mais simples ou uma estrutura mais complexa, mas redutível. Uma estrutura do

mesmo nível, mas mais simples, é considerada como tendo sido reduzida e, portanto, consumida.

Consideramos ainda que

> **uma transformação lógica que não consuma ou produza *redexes* não contribui para a resolução simbólica de expressões.**

De facto, o resultado de uma transformação lógica deve ser uma estrutura diferente daquela donde se partiu, mais simples ou que ainda possa ser reduzida. Caso contrário, a transformação não introduz nada de novo no processo de resolução.

Uma simplificação $\Gamma_S$ (Secção 2.5) é uma aplicação indirecta de outras transformações lógicas (reduções , conversões ou mesmo outras simplificações). As simplificações consumem ou produzem *redexes* através de reduções $\Gamma_R$ ou conversões $\Gamma_T$. A este tipo de reescrita demos o nome de reescrita por simplificação, ou simplesmente reescrita-$\downarrow$.

Se $s,t \in \mathcal{T}$ e $t[s]$ for um subtermo próprio de $t$, então uma simplificação de $t$, $\Gamma_S(t)$, é uma transformação do tipo $t[s] \to t[r\sigma]$, onde $s \to r\sigma$ é uma reescrita axiomática de $s$, $\Gamma_R(s)$ ou $\Gamma_T(s)$. A substituição $\sigma$ é simplesmente o unificador mais geral (Secção 3.3.2) de $s$ e o lado esquerdo $l$ de uma regra $l \to r \in \mathcal{B}$, da mesma classe, ou seja, $\sigma \equiv mgu(s,l)$. $t[r\sigma]$ é o resultado de uma composição lógica envolvendo o contexto $t[\bullet]$ e o termo $r\sigma$, ou seja, $[\lambda(t[\bullet])]\sigma$.

A reescrita-$\downarrow$ é aplicável a qualquer tipo de estrutura, ou seja,

$$\mathcal{C}^{\downarrow} = \{ f(s,t) \in \mathcal{T} \mid f \in \mathcal{O}p \wedge s,t \in \mathcal{T} \}$$

**Definição 4.3-5** ***Regra de Reescrita*-$\downarrow$**

> *Uma regra de reescrita-$\downarrow$ é uma regra de reescrita condicional, representada por $\downarrow^{N}\rightarrow$, que efectua transformações lógicas do tipo $s\rightarrow\left[\lambda_{\downarrow}(r)\right]\sigma$, ao nível $N$ de uma estrutura, onde $s\in\mathcal{C}^{\downarrow}$ e $\lambda_{\downarrow}(r)$ representa uma reescrita-$\downarrow$.*

Em teoria, qualquer regra de reescrita pode ser considerada uma regra de reescrita, por simplificação, aplicada ao nível $0$ da estrutura. De facto, uma regra de reescrita, qualquer, do tipo $\rightarrow$, não é mais do que uma regra de simplificação, do tipo $\downarrow^{0}\rightarrow$. Por exemplo, uma regra de reescrita-$\tau$ poderia ser representada por um símbolo, como $\downarrow^{0}\rightarrow_{\tau}$. Só que, neste caso, o símbolo $\downarrow^{0}$ seria redundante, pois não teria o significado operacional que, nesta tese, é atribuído a esse símbolo. Neste caso, o nível $0$ é considerado o argumento do seu próprio contexto.

As regras de reescrita-$\downarrow$ são regras de reescrita, condicionais, baseadas no predicado ***simplify***/11. Tal como todas as outras, também estas regras estão organizadas por classes $c\in\mathcal{C}^{\downarrow}$ e dispostas, também, por ordem decrescente, do seu poder redutor.

As regras de reescrita-$\downarrow$ são regras que designamos, também, por estratégicas. Existem quatro (4) tipos de estratégias:

- **Simplificação por cálculo** – A aplicação de uma regra de reescrita-$\kappa$ ao nível $N$ de uma estrutura, ou seja, uma estratégia do tipo

  $$\textbf{simplify}(L\rightarrow R, State, State, Rule)\leftarrow$$

  $$C_1,\ldots,C_n,$$

  $$\textbf{evaluate}(L\rightarrow R, State, Rule),$$

  $$C_{n+1},\ldots,C_m.$$

- **Simplificação por factorização** – A aplicação de uma regra de reescrita $\text{-}\gamma$ ao nível $N$ de uma estrutura, ou seja, uma estratégia do tipo

  **simplify**$(L \to R, State, State, Rule) \leftarrow$

  $C_1, \ldots, C_n,$

  **factorize**$(L \to R, State, Rule)$,

  $C_{n+1}, \ldots, C_m.$

- **Simplificação por recursividade** – A aplicação de uma regra de reescrita $\text{-}\downarrow$ ao nível $N$ de uma estrutura, ou seja, uma estratégia do tipo

  **simplify**$(L \to R, State, State, Rule) \leftarrow$

  $C_1, \ldots, C_n,$

  **simplify**$(L_1 \to R_1, State_1, State_2, Rule)$,

  $\lambda_{\downarrow}(R), C_{n+1}, \ldots, C_m.$

- **Simplificação por redução** ou **conversão axiomática**[44] – A aplicação de uma regra de reescrita $\text{-}\tau$, $\text{-}\alpha$ ou $\text{-}\psi$ ao nível $N$ de uma estrutura, ou seja, uma estratégia do tipo

  **simplify**$(L \to R, State, State, Rule) \leftarrow$

  $C_1, \ldots, C_n,$

  **apply_relation**$(L \to R, State, Rule)$,

  $C_{n+1}, \ldots, C_m.$

Como veremos, as reescritas $\text{-}\beta$ e $\text{-}\chi$ são transformações lógicas

[44] A diferença está no poder redutor da redução ou conversão (Secção 2.5).

contextualizadas por reescritas-$\alpha$ .

O poder redutor de uma regra de reescrita-$\downarrow$ é definido como se segue, em ordem decrescente:

- Reescrita-$\downarrow$ por cálculo ou redução axiomática, $\downarrow^{0}\rightarrow_{\kappa \text{ ou } \tau}$;
- Reescrita-$\downarrow$ por factorização, $\downarrow^{0}\rightarrow_{\gamma}$;
- Reescrita-$\downarrow$ por recursividade esquerda ou direita, $\downarrow^{N}\rightarrow_{\Gamma}$, com $N > 0$, onde $\Gamma$ representa uma transformação lógica qualquer (redução $\Gamma_{\boldsymbol{R}}$, conversão $\Gamma_{\boldsymbol{T}}$ ou simplificação $\Gamma_{\boldsymbol{S}}$);
- Reescrita-$\downarrow$ por conversão axiomática, $\downarrow^{N}\rightarrow_{\alpha \text{ ou } \psi}$ com $N \geq 0$.

Uma reescrita$-\downarrow$, $\lambda_{\downarrow}(R)$, é uma transformação lógica que resulta também da composição lógica de zero ou mais construtores-$\lambda$. O número de construtores utilizados é dado por $n-1$, onde $n$ é a profundidade onde teve lugar a reescrita axiomática.

**Exemplo 4.3-13** ***Reescrita*** $\downarrow$ *-log_root*

*No caso da reescrita* $\log_e(\sqrt{4})^3 \downarrow^{2}\rightarrow_{\beta} \log_e(4^{\frac{1}{2}})^3$, *a reescrita-*$\beta$ *teve lugar no nível 3 e foram necessários dois construtores-*$\lambda$, $\lambda$*-log e* $\lambda$*-pwr, para completar a reescrita.*

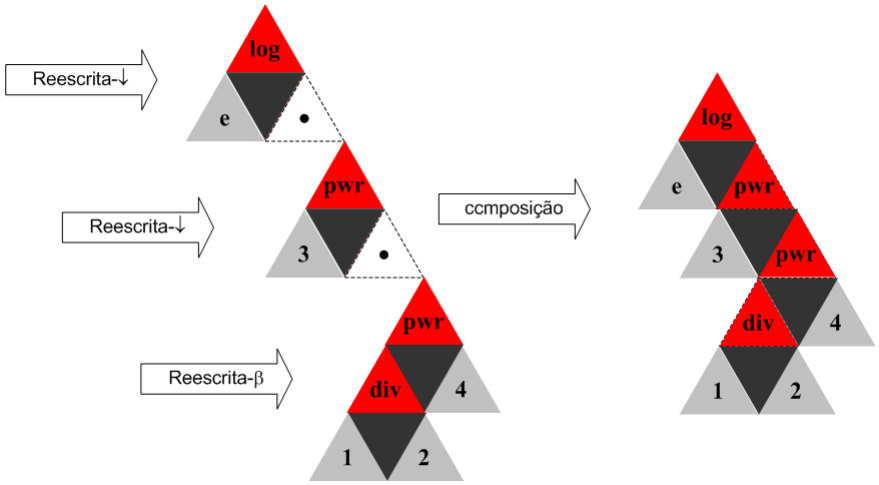



**Figura 4.3.6** *Simplificação e composição*

A reescrita-↓ reduz, ou converte, partes da estrutura e compõe de volta toda a estrutura.

Como veremos, mais adiante, as estratégias de resolução envolvem a aplicação de um certo número de regras de reescrita-↓. Cada operador matemático possui o seu próprio conjunto de regras estratégicas. Essas regras estão dispostas, sequencialmente, e constituem aquilo que poderíamos designar por plano estratégico do operador. Por exemplo, as reduções por cálculo são as primeiras a ser aplicadas no contexto desse plano.

Consideramos que

> **a aplicação de uma estratégia de resolução depende apenas da assinatura da expressão, ou seja, da estrutura triangular da sua componente principal.**

De facto, a aplicação de uma estratégia não é mais do que a selecção e a aplicação de uma regra de reescrita no contexto de um plano estratégico. A aplicação é feita ao nível das componentes e, portanto, depende apenas da assinaturas dessas componentes. Cada componente é considerada como a componente principal dessa parte da estrutura. No caso de uma expressão, com

uma assinatura *log_pwr* , estariam disponíveis duas estratégias distintas – uma para logaritmos e outra para potências. A estratégia preferencial dependeria, naturalmente, da presença de *redexes*, mas seria uma estratégia imposta pela estrutura logaritmica, ou seja, uma estratégia de redução, simplificação, ou conversão logaritmica.

#### *4.3.1.4 A Reescrita por Conversão axiomática*

Uma fórmula de cálculo é um tipo de fórmula que nos mereceu uma atenção especial. A fórmula de derivação da soma é um caso típico. Nesta tese, foram apenas estudadas as fórmulas de cálculo relacionadas com o operador derivada, ou seja, as regras de derivação.

Consideramos que

> **do ponto de vista do ensino da Matemática, ao nível do secundário, só faz sentido aplicar estas fórmulas no sentido convencional, ou seja, como fórmulas de decomposição.**

De facto, ao nível do ensino secundário, as regras de derivação são apenas aplicadas no sentido convencional, ou seja, como fórmulas de redução ou decomposição. Nesta tese, as fórmulas de cálculo relacionadas com o operador derivada são consideradas unidireccionais.

Consideremos, por exemplo, a fórmula de cálculo que define o cálculo da derivada de uma soma.

$$\frac{\mathrm{d}}{\mathrm{dx}}(f+g)=\frac{\mathrm{d}}{\mathrm{dx}}f+\frac{\mathrm{d}}{\mathrm{dx}}g$$

Uma forma de representar essa fórmula seria, por exemplo, através de duas asserções lógicas simétricas

$$\frac{\mathrm{d}}{\mathrm{dx}}(f+g) \rightarrow \frac{\mathrm{d}}{\mathrm{dx}} f + \frac{\mathrm{d}}{\mathrm{dx}} g \quad (1)$$

$$\frac{\mathrm{d}}{\mathrm{dx}} f + \frac{\mathrm{d}}{\mathrm{dx}} g \rightarrow \frac{\mathrm{d}}{\mathrm{dx}}(f+g) \quad (2)$$

O problema aqui não é essencialmente um problema de orientação, mas sim um problema de natureza semântica. De facto, a fórmula de derivação da soma define uma relação de equivalência entre os pares de operadores $(sum, der)$ e $(der, sum)$, ou seja, que "*a derivada de uma soma é equivalente a uma soma de derivadas, e vice-versa*". A primeira dessas asserções possui o significado de decomposição e a segunda, o de composição. Computacionalmente, tanto uma como a outra fariam sentido como regras de reescrita. Só que, no caso da decomposição, são sempre produzidos, um ou mais, *redexes* e no caso da composição, apenas quando existe uma afinidade operacional entre as componentes.

Por exemplo, no caso da expressão $\frac{\mathrm{d}}{\mathrm{dx}} 2x + \frac{\mathrm{d}}{\mathrm{dx}} 3x$, a composição faria sentido, do ponto de vista pedagógico, como forma de realçar a importância da simplificação no cálculo de derivadas. Só que o efeito, do ponto vista pedagógico, seria o mesmo se o problema fosse colocado como a resolução de $\frac{\mathrm{d}}{\mathrm{dx}}(2x+3x)$. Além disso, efectuar a transformação $\frac{\mathrm{d}}{\mathrm{dx}} 2x + \frac{\mathrm{d}}{\mathrm{dx}} 3x \rightarrow \frac{\mathrm{d}}{\mathrm{dx}}(2x+3x)$, não seria a estratégia preferencial, pois já existem dois *redexes* para resolver.

Optámos, portanto, pela orientação dessas fórmulas no sentido convencional, ou seja, como regras de decomposição. De facto, do ponto de vista do ensino, a decomposição é a forma convencional de se realizar esse cálculo. Resolver somas de derivadas não seriam expressões que, normalmente, se proporia a um aluno do secundário para resolver. Uma expressão desse tipo seria tratada

como uma soma e resolvida, simplificando as parcelas.

A importância de estratégias como "*simplificar antes de derivar*" pode ser evidenciada com exemplos, como a derivada $\frac{\mathrm{d}}{\mathrm{dx}}(2x+3x)$, chamando a atenção para a existência de *redexes*, nessa expressão. Essa é aliás uma das estratégias propostas, nesta tese, para a resolução desse tipo de expressões. A outra é "*decompor para simplificar*", ou seja, formar *redexes* para que se possa realizar o cálculo. Tanto uma como a outra são estratégias que contribuem para a redução do grau de complexidade dessas expressões. Com a decomposição são formadas expressões mais simples e com a simplificação do argumento, reduzida o grau de complexidade dessa expressão.

**Exemplo 4.3-14** ***Simplificar e derivar***

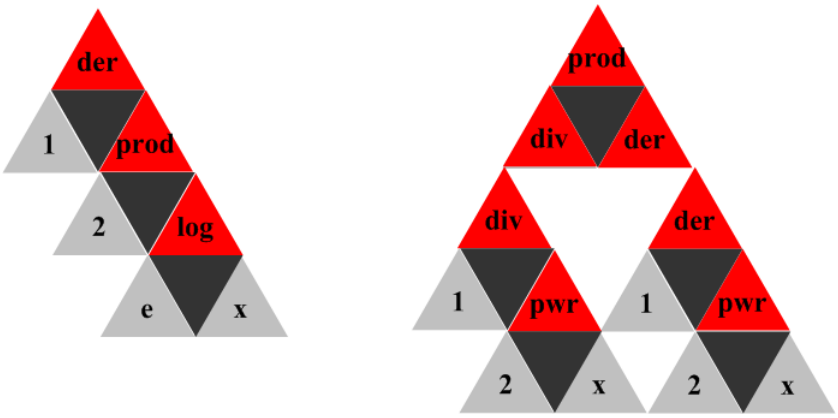
*No caso da expressão* $\frac{\mathrm{d}}{\mathrm{dx}}\log_e x^2$ *seria mais eficiente "simplificar e derivar" a expressão, ou seja, efectuar a transformação*

$$\frac{\mathrm{d}}{\mathrm{dx}}\log_e x^2 \downarrow\rightarrow \frac{\mathrm{d}}{\mathrm{dx}}(2\log_e x)$$

*do que "derivá-la e simplificá-la" , ou seja, efectuar a transformação*

$$\frac{\mathrm{d}}{\mathrm{dx}}\log_e x^2 \rightarrow \frac{1}{x^2}\frac{\mathrm{d}}{\mathrm{dx}}x^2$$

*O número de componentes seria, neste caso, maior.*




**a)** $\frac{\mathrm{d}}{\mathrm{dx}}(2\log_e x)$ **b)** $\frac{1}{x^2}\frac{\mathrm{d}}{\mathrm{dx}}x^2$

**Figura 4.3.7** *Estratégias de simplificação*

Orientadas neste sentido, as fórmulas de cálculo são, essencialmente, fórmulas de conversão, unidireccionais, entre tuplos de operadores. A este tipo de reescrita demos o nome de reescrita-$\psi$ .

Constatámos também que todas as fórmulas matemáticas, incluídas nesta tese, envolvem, no máximo, três operadores matemáticos, ou seja, que a sua estrutura pode ser representada por uma das cinco (5) estruturas triangulares básicas. Não podemos, no entanto, garantir que esse facto seja verificável com outras fórmulas matemáticas.

Uma grande parte das fórmulas de cálculo, incluídas nesta tese, possui um elevado poder redutor. Nesse grupo estão incluídas fórmulas, como

$$\frac{\mathrm{d}}{\mathrm{dx}}k \to 0\,,\ \frac{\mathrm{d}}{\mathrm{dx}}x \to 1\,,\ \frac{\mathrm{d}}{\mathrm{dx}}kx \to k\,,$$

$$\frac{\mathrm{d}}{\mathrm{dx}}x^k \to kx^{k-1} \text{ e } \frac{\mathrm{d}}{\mathrm{dx}}\log_e x \to \frac{1}{x}$$

Todas as outras são potencialmente redutoras.

Fórmulas, como $\frac{\mathrm{d}}{\mathrm{dx}}\sqrt[k]{x} \rightarrow \frac{\mathrm{d}}{\mathrm{dx}}x^{\frac{1}{k}}$ são potencialmente redutoras, pois são convertíveis noutras que sabemos ser redutoras, como por exemplo $\frac{\mathrm{d}}{\mathrm{dx}}x^{\frac{1}{k}} \rightarrow \frac{1}{k}x^{\frac{1}{k}-1}$.

O mesmo se poderia dizer de todas as outras. Todas elas potenciam a formação de um ou mais *redexes*.

A reescrita-$\psi$ é apenas aplicável a estruturas $\Xi$-*op* e $\Xi$-*op_op* onde $op \in \mathcal{S}ig_1^{\psi}$, $op\_op \in \mathcal{S}ig_2^{\psi}$ e

$$\mathcal{S}ig_1^{\psi} = \{der\}$$

$$\mathcal{S}ig_2^{\psi} = \{i\_j \in \mathcal{S}ig \mid i \in \mathcal{S}ig_1^{\psi} \wedge j \in \mathcal{O}p\}$$

$$\mathcal{C}_1^{\psi} = \{ f(s,t) \in \mathcal{T} \mid f \in \mathcal{S}ig_1^{\psi} \wedge s,t \in \mathcal{T} \}$$

$$\mathcal{C}_2^{\psi} = \{ f(s,g(t,u)) \in \mathcal{T} \mid f\_g \in \mathcal{S}ig_2^{\psi} \wedge s,t \in \mathcal{T} \}$$

$$\mathcal{C}^{\psi} = \mathcal{C}_1^{\psi} \cup \mathcal{C}_2^{\psi}$$

**Definição 4.3-6** ***Regra de Reescrita-$\psi$***

> *Uma regra de reescrita-$\psi$ é uma regra de reescrita condicional, representada por $\rightarrow_{\psi}$, que efectua transformações lógicas do tipo $s \rightarrow [\lambda_{\psi}(r)]\sigma$, onde $s \in \mathcal{C}^{\psi}$ e $\lambda_{\psi}(r)$ representa uma reescrita-$\psi$.*

As regras de reescrita-$\psi$ são regras de reescrita, condicionais, baseadas também no predicado ***apply_relation*** / 7. As regras de reescrita-$\psi$, estão organizadas por classes $c \in \mathcal{C}^{\psi}$ e dispostas também, por ordem decrescente, do seu poder redutor.

Uma reescrita-$\psi$, $\lambda_{\psi}(R)$, é uma transformação lógica, por conversão axiomática[45], que resulta também da composição lógica de um ou mais construtores -$\lambda$ ou da simples instanciação de uma variável lógica.

**Exemplo 4.3-15** ***Reescrita*** $\psi$*-der_sum*

*No caso da expressão* $\frac{\mathrm{d}}{\mathrm{dx}}(x+4)$*, a reescrita é designada por* $\psi$*-der_sum. Uma reescrita* $\psi$*-der_sum é representada por uma transformação do tipo*

$$\frac{\mathrm{d}}{\mathrm{dx}}(x+4) \rightarrow_{\psi} \frac{\mathrm{d}}{\mathrm{dx}}x + \frac{\mathrm{d}}{\mathrm{dx}}4$$

*A regra* $\psi$*-der_sum aplicada, neste caso, é uma regra do tipo*

**R1.** $\mathrm{der}(s, \mathrm{sum}(t, r)) \rightarrow \mathrm{sum}(\mathrm{der}(s, t), \mathrm{der}(s, r))$

***if*** $s \in \{1\} \wedge t \in \mathcal{T} \setminus \mathcal{I} \wedge r \in \mathcal{T} \setminus \{0\}$

---

A implementação Prolog de uma regra $\psi$***-der_sum*** está listado na Secção A.1.2 do Apêndice A.

Um outro tipo de fórmulas que nos mereceu também uma atenção especial,

[45] Note que numa conversão (Secção 2.5) o resultado não é necessariamente uma expressão mais simples.

foram as fórmulas matemáticas que devem, normalmente, ser orientadas em ambos os sentidos e que, portanto, designámos por bidireccionais. Nesse grupo estão incluídas as fórmulas que definem certas propriedades fundamentais das potências e dos logaritmos. Por exemplo, a relação fundamental entre raízes e potências, traduz de facto uma relação de equivalência entre esses dois tipos de operadores, que pode ser aplicada em ambos os sentidos.

As relações fundamentais entre operadores matemáticos definem, de facto, relações de equivalência entre $n$-tuplos de operadores. São fórmulas de conversão, bidireccionais, entre $n$-tuplos de operadores. Por exemplo, a relação fundamental entre logaritmos e potências pode ser encarada como uma fórmula de conversão, bidireccional, entre os pares de operadores $(log, pwr)$ e $(prod, log)$. Essa relação é traduzida pelo seguinte par de asserções lógicas

$$\log_a f^k \rightarrow k \log_a f \quad (1)$$

$$k \log_a f \rightarrow \log_a f^k \quad (2)$$

Os pares de operadores envolvidos, nessa relação, definem, de facto, duas classes de expressões, entre as quais existe uma relação de equivalência. O que essas duas asserções, (1) e (2), exprimem, é que "*o logaritmo de uma potência é convertível no produto de um logaritmo por uma constante, e vice-versa*". Mas como veremos, o poder redutor (Secção 2.4.2) dessa relação não é o mesmo em ambos os sentidos. A direcção com maior poder redutor é aquela que conduz à expressão mais simples, ou seja, a uma expressão com uma estrutura com menor grau de complexidade (Secção 2.4.2). No caso da relação entre logaritmos e potências, a primeira das asserções (1) é a que possui o maior poder redutor. A outra (2) é apenas uma forma de conversão axiomática, potencialmente redutora. Por exemplo, a transformação $2\log_e \sqrt{4} \rightarrow \log_e (\sqrt{4})^2$ seria potencialmente redutora.

As fórmulas bidireccionais fornecem, de facto, dois tipos de conversões (Secção 2.5). Uma, redutora e outra, potencialmente redutora. A este tipo de reescrita demos o nome de reescrita, por conversão axiomática, ou simplesmente, reescrita-$\alpha$ .

Mas, certas conversões, como veremos, só podem ter lugar em determinados contextos. Por exemplo, não faria sentido converter uma raiz a uma potência se dessa conversão não resultasse a formação de novos *redexes*. Assim, a conversão $\sqrt{3} \rightarrow 3^{\frac{1}{2}}$ não faria sentido, isoladamente, pois a raiz já se encontraria representada na sua forma mais simples. Porém, já o faria, se estivesse contextualizada por uma outra expressão como, por exemplo, $\log_e \sqrt{3}$ , pois, neste caso, a conversão potenciaria a formação de um novo *redex*. O contexto, neste caso, seria o operador logaritmo e o novo *redex*, a expressão $\log_e 3^{\frac{1}{2}}$ , que sabemos ser redutível. Para acomodar este tipo de transformações lógicas, foi necessário introduzir o conceito de reescrita contextualizada.

A reescrita-$\alpha$ é apenas aplicável a estruturas $\Xi$-*op_op* e $\Xi$-*op_op_op* , onde $op\_op \in \mathcal{S}ig_2^{\alpha}$ , $op\_op\_op \in \mathcal{S}ig_3^{\alpha}$ e

$$\mathcal{S}ig_1^{\alpha} = \{sum, diff, prod, div, pwr, root, log, exp\}$$

$$\mathcal{S}ig_2^{\alpha\text{-}1} = \{i\_j \in \mathcal{S}ig \mid i \in \{sum, diff\} \wedge j \in \mathcal{S}ig_1^{\alpha} \setminus \{log, exp\}\}$$

$$\mathcal{S}ig_2^{\alpha\text{-}2} = \{i\_j \in \mathcal{S}ig \mid i \in \{prod, div\} \wedge j \in \mathcal{S}ig_1^{\alpha} \setminus \{log, exp\}\}$$

$$\mathcal{S}ig_2^{\alpha\text{-}3} = \{i\_j \in \mathcal{S}ig \mid i, j \in \{pwr, root\}\}$$

$$\mathcal{S}ig_2^{\alpha\text{-}4} = \{i\_j \in \mathcal{S}ig \mid i \in \{log\} \wedge i, j \in \mathcal{S}ig_1^{\alpha} \setminus \{sum, diff, log\}\}$$

$$\mathcal{S}ig_2^{\alpha\text{-}5} = \left\{ i\_j \in \mathcal{S}ig \mid i \in \{exp\} \wedge i, j \in \mathcal{S}ig_1^{\alpha} \setminus \{pwr, root, exp\} \right\}$$

$$\mathcal{S}ig_3^{\alpha\text{-}1} = \left\{ i\_j\_k \in \mathcal{S}ig \mid i\_j \in \mathcal{S}ig_2^{\alpha\text{-}sum} \wedge k \in \{sum, diff, prod, div\} \right\}$$

$$\mathcal{S}ig_3^{\alpha\text{-}2} = \left\{ i\_j\_k \in \mathcal{S}ig \mid i\_j \in \mathcal{S}ig_2^{\alpha\text{-}prod} \wedge k \in \{prod, div\} \right\}$$

$$\mathcal{S}ig_3^{\alpha\text{-}3} = \left\{ i\_j\_k \in \mathcal{S}ig \mid i\_j \in \mathcal{S}ig_2^{\alpha\text{-}div} \wedge k \in \{sum, diff, prod, div\} \right\}$$

$$\mathcal{S}ig_2^{\alpha} = \bigcup_{i=1}^{5} \mathcal{S}ig_2^{\alpha\text{-}i}$$

$$\mathcal{S}ig_3^{\alpha} = \bigcup_{i=1}^{3} \mathcal{S}ig_3^{\alpha\text{-}i}$$

$$\mathcal{C}_2^{\alpha} = \left\{ f(s, g(t,u)), f(g(s,t),u) \in \mathcal{T} \mid f\_g \in \mathcal{S}ig_2^{\alpha} \wedge s,t,u \in \mathcal{T} \right\}$$

$$\mathcal{C}_3^{\alpha} = \left\{ f(g(s,t), h(u,v)) \in \mathcal{T} \mid f\_g\_h \in \mathcal{S}ig_3^{\alpha} \wedge s,t,u,v \in \mathcal{T} \right\}$$

$$\mathcal{C}^{\alpha} = \mathcal{C}_2^{\alpha} \cup \mathcal{C}_3^{\alpha}$$

**Definição 4.3-7** ***Regra de Reescrita-$\alpha$***

> *Uma regra de reescrita-$\alpha$ é uma regra de reescrita condicional, representada por $\rightarrow_{\alpha}$, que efectua transformações lógicas do tipo $s \rightarrow [\lambda_{\alpha}(r)]\sigma$, onde $s \in \mathcal{C}^{\alpha}$ e $\lambda_{\alpha}(r)$ representa uma reescrita-$\alpha$.*

As regras de reescrita-$\alpha$ são regras de reescrita, condicionais, baseadas também no predicado ***apply_relation*** / 7. Estas regras estão também organizadas por classes $c \in \mathcal{C}^{\alpha}$ e dispostas por ordem decrescente do seu poder redutor.

Uma reescrita-$\alpha$, $\lambda_\alpha(R)$, é uma transformação lógica, por conversão axiomática, redutora, ou potencialmente redutora, que resulta da composição lógica de um ou mais construtores-$\lambda$.

**Exemplo 4.3-16** ***Reescrita*** $\alpha$*-log_pwr*

*Uma possível reescrita da expressão* $\log_e 4^2$ *é a expressão* $2\log_e 4$*, ou seja, a transformação lógica*

$$\log_e 4^2 \to_\alpha 2\log_e 4$$

*A regra* $\alpha$*-log_pwr aplicada nessa transformação é uma regra de reescrita do tipo*

**R1.** $\log(s, \text{pwr}(t, r)) \to \text{prod}(t, \log(s, r))$

***if*** $s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{-1,0,1\} \wedge r \in \mathcal{F} \cup \{x\}$

---

A implementação Prolog de uma regra $\alpha$***-log_pwr*** está listado na Secção A.1.2 do Apêndice A.

**Exemplo 4.3-17** ***Resolução simbólica de*** $\log_e 4^2$

*Uma possível solução seria a seguinte cadeia de transformações*

$$\log_e 4^2 \to_\alpha 2\log_e 4 \downarrow^1 \to_\gamma 2\log_e 2^2 \downarrow^1 \to_\alpha 2(2\log_e 2) \to_\alpha$$

$$\to_\alpha 2(2)\log_e 2 \downarrow^1 \to_\kappa 4\log_e 2$$

*A sua base axiomática seria a seguinte:*

$$\mathcal{B}(s) = \{ l_i \to r_j \in \mathcal{B} \mid i \neq j \wedge i, j \in \mathcal{R}w(s) \}$$

*onde*

$$\mathcal{R}w(s) = \{ log, log\_pwr, prod, prod\_log, pwr, pwr\_pwr \}$$

*e*

**R1.** $\log(s, \text{pwr}(t, r)) \to \text{prod}(t, \log(s, r))$

$$\textit{if } s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{-1,0,1\} \wedge r \in \mathcal{F} \cup \{x\}$$

**R2.** $\log(s, t) \to \log(s, r)$

$$\textit{if } s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{0,1\} \wedge r = \textit{factor\_out}(t)$$

**R3.** $\text{prod}(s, \text{prod}(t, r)) \to \text{prod}(\text{prod}(s, t), r)$

$$\textit{if } s, t \in \mathcal{T} \setminus \{-1,0,1\} \wedge r \notin \mathbb{N} \wedge \textit{akin}(s, t)$$

**R4.** $\text{prod}(s, t) \to r$

$$\textit{if } s, t \in \mathbb{N} \wedge r = s \cdot t$$

---

A reescrita contextualizada é um caso especial de reescrita, que envolve apenas um número restrito de fórmulas e que tem a ver, essencialmente, com a reescrita de padrões potencialmente redutores, quando devidamente contextualizados. A aplicação de reescritas-$\psi$, no sentido inverso, pode ser potencialmente redutora, se tratada de forma contextualizada. Essa hipótese,

porém, não foi contemplada, nesta tese, por razões que já foram invocadas.

A reescrita contextualizada utiliza outras regras como contexto. Por exemplo, uma expressão, como $\log_e \sqrt{3}$, com uma assinatura *log_root*, pode ser convertida, por reescrita contextualizada, numa expressão equivalente, $\log_e 3^{\frac{1}{2}}$, com a assinatura *log_div_pwr*. A raiz é convertida no contexto de uma regra $\alpha$-*log_root*.

Dershowitz et al. [DER01], consideram que regras orientadas em ambos os sentidos podem causar problemas de terminação. Esse problema é, normalmente, causado pela perpetuação de padrões. A sua eliminação pode ser conseguida com a aplicação do conceito de reescrita contextualizada, pois, com esse tipo de reescrita, só são formados novos padrões, potencialmente, redutores. A conversão entre raízes e potências é um caso típico desse tipo reescrita. Por exemplo, a contextualização de uma das reescritas garante que a conversão só é efectuada se houver garantia de formação de novos *redexes*. Caso contrário, a reescrita é rejeitada. No caso da conversão de raízes, essa decisão é tomada com base no resultado da factorização do radicando.

Para esta tese, só foi considerada a factorização de números inteiros. A esse tipo reescrita demos o nome de reescrita contextualizada, por conversão axiomática, ou simplesmente, reescrita-$\beta$. Numa reescrita-$\beta$, o contexto de uma expressão é definido pela estrutura triangular que a insere. Por exemplo, uma estrutura como $\Xi$-*log_root*, define o contexto da sua componente *root*. Uma expressão só pode estar contextualizada, se for componente de uma outra.

Assim, consideramos que

> **a assinatura de uma expressão define o contexto dos seus argumentos.**

A reescrita-$\beta$ é apenas aplicável a estruturas $\Xi$-*op* devidamente contextualizadas por estruturas $\Xi$-*op_op* onde $op \in \mathcal{Sig}_1^{\beta}$, $op\_op \in \mathcal{Sig}_2^{\beta}$ e

$$\mathcal{Sig}_1^{\beta} = \{root, log, exp\}$$

$$\mathcal{Sig}_2^{\beta\text{-}root} = \{i\_j \in \mathcal{Sig} \mid i \in \{pwr, root, log\} \wedge j \in \{root\}\}$$

$$\mathcal{Sig}_2^{\beta\text{-}log} = \{i\_j \in \mathcal{Sig} \mid i \in \{log\} \wedge j \in \{exp\}\}$$

$$\mathcal{Sig}_2^{\beta\text{-}exp} = \{i\_j \in \mathcal{Sig} \mid i \in \{exp\} \wedge j \in \{log\}\}$$

$$\mathcal{Sig}_2^{\beta} = \mathcal{Sig}_2^{\beta\text{-}root} \cup \mathcal{Sig}_2^{\beta\text{-}log} \cup \mathcal{Sig}_2^{\beta\text{-}exp}$$

$$\mathcal{C}^{\beta} = \{ f(s,t) \in \mathcal{T} \mid f \in \mathcal{Sig}_1^{\beta} \wedge g(u, f(s,t)) \in \mathcal{T} \wedge g\_f \in \mathcal{Sig}_2^{\beta} \wedge s,t,u \in \mathcal{T} \}$$

**Definição 4.3-8** ***Regra de Reescrita-$\beta$***

> *Uma regra de reescrita-$\beta$ é uma regra de reescrita condicional, representada por $\rightarrow_{\beta}$, que efectua transformações lógicas do tipo $s \rightarrow [\lambda_{\beta}(r)]\sigma$, onde $s \in \mathcal{C}^{\beta}$ e $\lambda_{\beta}(r)$ representa uma reescrita-$\beta$.*

As regras de reescrita-$\beta$ são regras de reescrita, condicionais, baseadas no predicado ***convert*** / 7. As regras estão também organizadas por classes $c \in \mathcal{C}^{\beta}$ e dispostas por ordem decrescente do seu poder redutor.

A aplicação de uma regra de reescrita-$\beta$ pode ser representada simbolicamente por uma expressão do tipo

**apply_relation**$(L \rightarrow \mathrm{R}, State, Rule) \leftarrow$

$\mathrm{C}_1, \ldots, \mathrm{C}_n,$

**convert**$(L \rightarrow \mathrm{R}, State, Rule)$,

$\lambda_\beta(R)$.

**convert**$(L \rightarrow \mathrm{R}, State, Rule)$,

$\mathrm{C}_1, \ldots, \mathrm{C}_n$,

$\lambda_\beta(R)$.

Uma reescrita-$\beta$, $\lambda_\beta(R)$, é uma reescrita contextualizada, por conversão axiomática, potencialmente redutora, que resulta da composição lógica de um ou mais construtores-$\lambda$.

A regra de reescrita que serve de contexto a uma reescrita-$\beta$ é designada também por regra de reescrita-$\beta$. É, precisamente, a assinatura dessa regra que define o contexto da reescrita-$\beta$. Por exemplo, uma reescrita $\beta$-*root* só pode ser aplicada se estiver devidamente contextualizada por uma reescrita, como $\beta$-*log_root*.

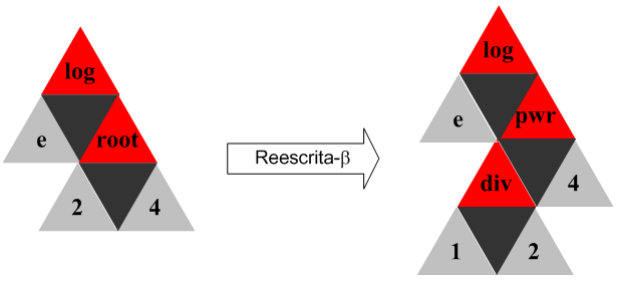



**Figura 4.3.8** *Reescrita contextualizada*

**Exemplo 4.3-18** *Reescrita* $\beta$-*log_root*

*No caso da expressão* $\log_e \sqrt{4}$, *com uma assinatura* log_root, *a raiz pode ser convertida por reescrita contextualizada, aplicando-lhe uma regra do tipo*

**R1.** $\log(s, \text{root}(t, r)) \to \log(s, q)$

$$\textit{if } s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{0,1\} \wedge r \in \mathcal{T}^+ \cup \{0,1\} \wedge$$

$$q = \textit{convert}(\text{root}(t, r))$$

**R2.** $\text{root}(s, t) \to \text{pwr}(\text{div}(1, s), t)$

$$\textit{if } s \in \mathbb{N} \setminus \{0,1\} \wedge t \in \mathcal{T}$$

---

**Exemplo 4.3-19** ***Resolução simbólica de*** $\log_e \sqrt{4}$

*Uma possível solução seria a seguinte cadeia de transformações*[46]

$$\log_e \sqrt{4} \to_\beta \log_e 4^{\frac{1}{2}} \to_\alpha \frac{1}{2} \log_e 4 \downarrow^1 \to_\gamma \frac{1}{2} \log_e 2^2 \downarrow^1 \to_\alpha$$

$$\downarrow^1 \to_\alpha \frac{1}{2}(2) \log_e 2 \downarrow^1 \to_\alpha \frac{1}{2} 2 \log_e 2 \downarrow^1 \to_\alpha 1 \log_e 2 \to_\alpha \log_e 2$$

*A base axiomática seria, neste caso, a seguinte:*

$$\mathcal{B}(s) = \left\{ l_i \to r_j \in \mathcal{B} \mid i \neq j \wedge i, j \in \mathcal{R}w(s) \right\}$$

[46] A reescrita -$\gamma$ é também uma reescrita contextualizada que abordaremos mais adiante.

*onde*

$$\mathcal{R}w(s) = \{ prod, prod\_prod, pwr, pwr\_pwr, root, root\_pwr, log, log\_pwr, log\_root \}$$

*e*

**R1.** $\log(s, \text{root}(t, r)) \rightarrow \log(s, q)$

$$\textit{if } s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{0,1\} \wedge r \in \mathcal{T}^{+} \cup \{0,1\} \wedge q = convert(\text{root}(t, r))$$

**R2.** $\text{root}(s, t) \rightarrow \text{pwr}(\text{div}(1, s), t)$

$$\textit{if } s \in \mathbb{N} \setminus \{0,1\} \wedge t \in \mathcal{T}$$

**R3.** $\log(s, \text{pwr}(t, r)) \rightarrow \text{prod}(t, \log(s, r))$

$$\textit{if } s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge fraction(t) \wedge r \in \mathbb{N} \cup \mathcal{K} \setminus \{-1,0,1\}$$

**R4.** $\log(s, t) \rightarrow \log(s, r)$

$$\textit{if } s \in \mathbb{N} \cup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{0,1\} \wedge r = factor\_out(t)$$

**R5.** $\text{prod}(s, \text{prod}(t, r)) \rightarrow \text{prod}(\text{prod}(s, t), r)$

$$\textit{if } s, t \in \mathcal{T} \setminus \{-1,0,1\} \wedge r \notin \mathbb{N} \wedge akin(s, t)$$

**R6.** $\text{prod}(s, t) \rightarrow 1$

***if*** $s,t \in \mathcal{T} \setminus \{-1,0,1\} \wedge symmetric(s,t)$

**R7.** $\text{prod}(s,t) \rightarrow t$

***if*** $s \in \{1\} \wedge t \in \mathcal{T} \setminus \{-1,0,1\}$

---

A implementação Prolog de uma regra $\beta$-***root*** está listada na Secção A.1.4 do Apêndice A e a de uma regra $\beta$-***log_root*** na Secção A.1.2.

Mas como já referimos atrás, não basta que a reescrita esteja devidamente contextualizada. É necessário que a reescrita potencie também a formação de novos *redexes*. Por exemplo, a expressão $\sqrt[3]{x^2}$ não é convertível, por reescrita contextualizada $\beta$-*root_pwr*, pois o grau da raiz é, neste caso, superior ao da potência e, portanto, não potencia a formação de novos *redexes*.

Como as regras de reescrita são condicionais, pode haver regras com a mesma assinatura, que diferem apenas nas condicionantes que impõem. Essas variantes estão dispostas, por ordem decrescente, do seu poder redutor. Assim, se uma expressão satisfaz a mais do que uma regra, a mais redutora é a que é aplicada primeiro. No caso de regras, com assinaturas *root_pwr*, a base axiomática contém variantes como

**R1.** $\text{root}(s,\text{pwr}(t,r)) \rightarrow r$

***if*** $s \in \mathbb{N} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{-1,0,1\} \wedge s = t$

**R2.** $\text{root}(s,\text{pwr}(t,r)) \rightarrow q$

***if*** $s \in \mathbb{N} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{-1,0,1\} \wedge r \in \mathbb{N} \bigcup \{x\} \wedge s > t \wedge$

$$q = convert(\text{root}(t, r))$$

**R3.** $\text{root}(s, \text{pwr}(t, r)) \to q$

$$\textbf{\textit{if}}\ s \in \mathbb{N} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{-1,0,1\} \wedge r \in \mathbb{N} \cup \{x\} \wedge s < t \wedge$$

$$q = convert(\text{root}(s, \text{pwr}(t, r)))$$

É preciso notar que a reescrita-$\beta$ não é a única reescrita contextualizada. A factorização de números inteiros positivos, que mencionámos atrás, é também uma forma de reescrita contextualizada. Tal como no caso das raízes, que abordámos atrás, só faz sentido factorizar números inteiros que estejam devidamente contextualizados e que potenciem, portanto, a formação de novos *redexes*. O que significa, naturalmente, que só determinados contextos poderão beneficiar deste tipo de reescrita. A este tipo de reescrita demos o nome de reescrita contextualizada, por factorização, ou simplesmente reescrita-$\gamma$.

A reescrita-$\gamma$ é apenas aplicável a estruturas $\Xi$-*none* devidamente contextualizadas por estruturas $\Xi$-*op* onde $op \in \mathcal{S}ig_1^{\gamma}$ e

$$\mathcal{S}ig_1^{\gamma} = \{pwr, root, log, exp\}$$

$$\mathcal{C}^{\gamma} = \left\{ t \in \mathbb{N} \mid f(s,t) \in \mathcal{T} \wedge f \in \mathcal{S}ig_1^{\gamma} \wedge s \in \mathbb{N} \setminus \{0,1\} \cup \{e\} \right\}\ s,t,u \in \mathcal{T} \}$$

**Definição 4.3-9** ***Regra de Reescrita-$\gamma$***

> *Uma regra de reescrita-$\gamma$ é uma regra de reescrita condicional, representada por $\to_{\gamma}$, que efectua transformações lógicas do tipo $s \to \left[\lambda_{\gamma}(r)\right]\sigma$, onde $s \in \mathcal{C}^{\gamma}$ e $\lambda_{\gamma}(r)$ representa uma reescrita-$\gamma$.*

As regras de reescrita-$\gamma$ são regras de reescrita, condicionais, baseadas no predicado ***factorize*** / 7 . Tal como todas as outras, também estas estão organizadas por classes $c \in \mathcal{C}^{\gamma}$ e dispostas, por ordem decrescente, do seu poder redutor.

A aplicação de uma regra de reescrita-$\gamma$ pode ser representada simbolicamente por uma expressão do tipo

$$\textbf{factorize}(L \to \mathrm{R}, State, Rule) \leftarrow$$

$$\mathrm{C}_1, \ldots, \mathrm{C}_n,$$

$$\lambda_{\gamma}(R).$$

Uma reescrita-$\gamma$ , $\lambda_{\gamma}(R)$ , é uma reescrita contextualizada por factorização, potencialmente redutora, que resulta da composição lógica de um ou mais construtores-$\lambda$ .

**Exemplo 4.3-20** ***Reescrita*** $\gamma$***-root***

*A expressão* $\sqrt{16}$*, com uma estrutura de tipo* $\Xi$*-root , é redutível por reescrita-*$\gamma$ *, aplicando-lhe uma regra do tipo*

**R1.** $\mathrm{root}(s,t) \to \mathrm{root}(s,r)$

$$\textit{if } s,t \in \mathbb{N} \setminus \{0,1\} \wedge r = \mathit{factor\_out}(t)$$

---

**Exemplo 4.3-21** ***Resolução simbólica de*** $\sqrt{16}$

*Uma possível solução é a seguinte cadeia de transformações*

$$\underbrace{\sqrt{16}}_{root}\downarrow^{1}\rightarrow_{\gamma}\underbrace{\sqrt{2^{4}}}_{root\_pwr}\rightarrow_{\beta}\underbrace{(2^{4})^{\frac{1}{2}}}_{pwr\_pwr}\rightarrow_{\alpha}2\underbrace{4^{\left(\frac{1}{2}\right)}}_{prod\_div}\downarrow^{1}\rightarrow_{\tau}2\underbrace{\frac{4}{2}}_{div}\downarrow^{1}\rightarrow_{\tau}\underbrace{2^{2}}_{pwr}\rightarrow_{\tau}\underbrace{4}_{none}$$

*A base axiomática seria, neste caso, a seguinte:*

$$\mathcal{B}(s)=\left\{l_i\rightarrow r_j\in\mathcal{B}\mid i\neq j\wedge i,j\in\mathcal{R}w(s)\right\}$$

*onde*

$$\mathcal{R}w(s)=\{\,prod,\,prod\_div,\,div,\,pwr,\,pwr\_pwr,$$

$$root,root\_pwr\}$$

*e*

**R1.** $\text{root}(s,t)\rightarrow\text{root}(s,r)$

$$\textit{if}\ s,t\in\mathbb{N}\setminus\{0,1\}\wedge r=\ factor\_out(t)$$

**R2.** $\text{root}(s,\text{pwr}(t,r))\rightarrow q$

$$\textit{if}\ s,t,r\in\mathbb{N}\setminus\{0,1\}\wedge s>t\wedge$$

$$q=convert(\text{root}(s,\text{pwr}(t,r)))$$

**R3.** $\text{pwr}(s,\text{pwr}(t,r))\rightarrow\text{pwr}(\text{prod}(s,t),r)$

$$\textit{if}\ s\in\mathbb{N}\cup\mathcal{K}\setminus\{-1,0,1\}\wedge t,r\in\mathbb{N}\setminus\{0,1\}$$

**R4.** $\text{prod}(s,\text{div}(t,r))\rightarrow\text{div}(q,t)$

$$\textit{if } s,t,r \in \mathbb{N} \setminus \{0,1\} \wedge q = \textit{eval}(\text{prod}(s,t))$$

**R5.** $\text{div}(s,t) \rightarrow r$

$$\textit{if } s,t \in \mathbb{N} \setminus \{0,1\} \wedge s \neq t \wedge \textit{multiple}(s,t) \wedge r = s \div t$$

**R6.** $\text{pwr}(s,t) \rightarrow r$

$$\textit{if } s,t \in \mathbb{N} \setminus \{0,1\} \wedge r = s \hat{} t$$

*Como se pode ver, a reescrita contextualizada $\gamma$-root torna possível a aplicação de uma reescrita $\beta$-root_pwr. A contextualização da regra $\gamma$-root está implícita na própria regra.*

---

A implementação Prolog de uma regra $\gamma$-***root*** está listada na Secção A.1.4 do Apêndice A.

O facto da estrutura triangular ser uma estrutura arbórea e binária, torna possível a determinação de afinidades operacionais entre os seus vários operadores e argumentos, através de uma análise recursiva da sua estrutura triangular. Com base na afinidade operacional é possível decidir sobre a aplicabilidade de certas reescritas, como por exemplo, a aplicação da propriedade associativa. A aproximação de termos afins potencia a formação de *redexes*. Esse tipo de aproximação é o que Bundy [BUN85] chama de *atracção* e que potencia a *colecta* de termos afins.

Consideramos, portanto, que

> **a afinidade operacional é uma forma simples mas bastante eficaz de potenciar a formação de *redexes*.**

A afinidade operacional é baseada no predicado ***akin*** / 3 .

**Definição 4.3-10** *Afinidade Operacional*

> *Dada uma estrutura* $f(a,b)$, *diz-se que* $a$ *e* $b$ *são, operacionalmente afins, e escreve-se* $akin(a,b)$, *se* $f(a,b)$ *for potenciadora de novos redexes.*

Sejam $f(s,t)$, $f(g(s,t),u)$, $f(u,g(s,t))$ e $f(g(s,t),h(u,v))$, termos com estruturas triangulares. Diz-se que esses termos são potenciadores de novos *redexes*, se

- $f \in \mathcal{O}p \wedge s = t$
- $f, g \in \mathcal{O}p^{sum} \wedge akin(s,u) \vee akin(t,u)$
- $f \in \mathcal{O}p^{sum} \wedge g \in \mathcal{O}p^{prod} \wedge akin(s,u) \vee akin(t,u)$
- $f, g, h \in \mathcal{O}p^{sum} \wedge akin(s,u) \vee akin(t,v)$
- $f, g \in \mathcal{O}p^{sum} \wedge h \in \mathcal{O}p^{prod} \wedge akin(s,v) \vee akin(t,u)$
- $f \in \mathcal{O}p^{sum} \wedge g, h \in \mathcal{O}p^{prod} \wedge akin(s,u) \vee akin(t,v)$
- $f \in \mathcal{O}p^{sum} \wedge g, h \in \{log\} \wedge s = u \wedge akin(t,v)$
- $f, g \in \mathcal{O}p^{prod} \wedge akin(s,u) \vee akin(t,u)$
- $f, g \in \mathcal{O}p^{prod} \wedge h \in \mathcal{O}p^{pwr} \wedge s = v \vee t = v$
- $f \in \mathcal{O}p^{prod} \wedge g \in \mathcal{O}p^{pwr} \wedge s = u \vee t = u$

- $f, h \in \mathcal{O}p^{prod} \wedge g \in \mathcal{O}p^{pwr} \wedge t = u \vee t = v$
- $f \in \mathcal{O}p^{prod} \wedge g, h \in \mathcal{O}p^{pwr} \wedge (s = u \wedge akin(t, v)) \vee t = v$
- $f \in \mathcal{O}p^{prod} \wedge g \in \mathcal{O}p^{pwr} \wedge h \in \mathcal{O}p^{root} \wedge t = u$
- $f \in \mathcal{O}p^{prod} \wedge g \in \mathcal{O}p^{root} \wedge h \in \mathcal{O}p^{pwr} \wedge t = u$
- $f \in \mathcal{O}p^{prod} \wedge g, h \in \mathcal{O}p^{root} \wedge (s = u \wedge akin(t, v)) \vee t = v$
- $f \in \mathcal{O}p^{prod} \wedge g, h \in \{exp\} \wedge s = u \wedge akin(t, v) \vee t = v$

**Exemplo 4.3-22** ***Afinidades operacionais***

*A expressão* $(\sqrt{x} + x) + 2x$ *é potenciadora de novos redexes, pois existe uma afinidade operacional entre os termos* $\sqrt{x} + x$ *e* $2x$*. Essa afinidade propicia a formação do redex* $x + 2x$*, por associatividade. De facto, essa é a única afinidade operacional existente entre esses dois termos. No contexto dessa soma, não existe qualquer afinidade entre os termos* $\sqrt{x}$ *e* $2x$*.*

---

Consideramos que

> **a afinidade operacional deve ser um requisito essencial no caso da associatividade.**

De facto, só faz sentido associar termos afins, ou seja, se dessa associação resultar a formação de um ou mais *redexes*. Como já o afirmámos várias vezes, um dos principais objectivos de uma reescrita é, precisamente, a eliminação de *redexes* ou a formação de novos *redexes*. Por exemplo, no caso da expressão $(x+1)+2$, faria todo o sentido associar os elementos $1$ e $2$, pois dessa associação resultaria a

formação do *redex* $1+2$. Já o mesmo não sucederia com a expressão $(x+\sqrt{x})+2$.

A implementação Prolog do predicado ***akin*** / 3 está listada na Secção A.3.1 do Apêndice A.

### *4.3.1.5 A Reescrita por Composição Lógica*

As regras de formação de operadores são regras de reescrita contextualizada, designadas por $\lambda$-*op*, utilizadas na composição lógica de expressões matemáticas, por construção-$\lambda$. A composição lógica de expressões matemáticas é um tipo de reescrita baseada numa forma simplificada da composição semântica de Montague, inspirada no cálculo lambda, mas que utiliza apenas lógicas de 1ª ordem [PER87].

Uma expressão, como $\log_e \sqrt{4}$ é formada pela composição lógica de dois termos livres, designados por construtores-$\lambda$ (Secção 4.2.2). Os construtores-$\lambda$ são termos livres, designados por $\lambda$-*op*, que formam os termos funcionais da sua classe, por unificação (Secção 3.3.2). O termo livre $\log(e, Root, Log)$ representa um construtor-$\lambda$, da classe *log*, que denotamos por $\lambda$-*log*.

Por composição lógica, esse termo e o construtor $\lambda$-*root*, $\text{root}(2,4,Root)$ formam o termo funcional $\log(e, \text{root}(2,4))$, com uma assinatura *log_root*, ou seja, a representação lógica da expressão $\log_e \sqrt{4}$. A este tipo de reescrita demos o nome de reescrita, por composição lógica, ou simplesmente reescrita-$\lambda$.

**Definição 4.3-11** ***Regra de Reescrita*** *-$\lambda$*

> *Uma regra de reescrita-$\lambda$ é uma regra de reescrita, não condicional, representada por $\rightarrow_\lambda$, que efectua transformações lógicas do tipo $\rightarrow[\lambda(r)]\sigma$, onde $\lambda(r)$ representa uma reescrita-$\lambda$.*

As regras de reescrita-$\lambda$ são regras de reescrita, contextualizadas, baseadas nos predicados $\boldsymbol{op}/3$ onde $op \in \mathcal{O}p$. As regras de reescrita-$\lambda$ estão organizadas por classes $c \in \mathcal{C}^{\lambda}$ onde $\mathcal{C}^{\lambda} = \mathcal{O}p$.

A aplicação de uma regra de reescrita-$\lambda$ pode ser representada simbolicamente por uma expressão do tipo

$$\boldsymbol{op}(\rightarrow \lambda(R)) \leftarrow$$

onde $op \in \mathcal{O}p$.

**Exemplo 4.3-23** ***Composição lógica*** *$\lambda$-log_root*

*A expressão* $\log_e \sqrt{4}$ *é o resultado da composição lógica de dois construtores* -$\lambda$*:* $\log(Arg_1, Root, Log)$ *e* $\text{root}(Arg_{21}, Arg_{22}, Root)$. *As variáveis lógicas* $Arg_i$ *são instanciadas por* $Arg_1 = e$, $Arg_{21} = 2$ *e* $Arg_{22} = 4$.

*A formação dessa expressão pode ser representada, de forma simbólica, pela seguinte cadeia de transformações lógicas*

$$(Log)_{Log=\log_e G} \rightarrow_{\lambda} (\log_e Root)_{Root=\sqrt{4}} \rightarrow_{\lambda} \log_e \sqrt{4}$$

---

A implementação Prolog de uma regra $\lambda$-***log*** está listada na Secção A.1.5 do Apêndice A.

### *4.3.1.6 A Reescrita por comutação*

A comutatividade de certas operações pode introduzir dificuldades ao processo de reescrita [BEE98, DER01].

Consideramos que

> **é necessário optar por uma *ordem convencional de escrita* de expressões para resolver o problema [SOL82].**

De facto, a opção por uma ordem convencional de escrita permite que a comutação possa ser feita de forma automática e apenas num único sentido. Para esta tese foram optados os seguintes critérios gerais para a escrita para somas, subtracções e produtos:

- Numa soma ou subtracção, as constantes, numéricas ou simbólicas, são mantidas à direita de qualquer expressão, não numérica. Por exemplo, $x+4$, $\sqrt{x}-5$ ou $x-\sqrt{2}$;
- Numa soma ou subtracção, as constantes numéricas são mantidas à direita de qualquer constante simbólica. Por exemplo, $\sqrt{3}+5$ ou $\log_e 3-7$;
- Num produto, as constantes, numéricas ou simbólicas, são mantidas à esquerda de qualquer expressão, não numérica. Por exemplo, $2\sqrt{x}$, $\sqrt{2}\sqrt{x}$ ou $\log_{10} 2(x)$;
- Num produto, as constantes numéricas são mantidas à esquerda de qualquer constante simbólica. Por exemplo, $3\log_e 5$ ou $5\sqrt{3}$.

Assim, expressões como $2+x$ e $x(2)$, são reescritas, por comutação, nas expressões equivalentes, $x+2$ e $2x$, respectivamente. No caso de subtracções, a comutação é efectuada depois de o sinal ter sido posto em evidência. Uma expressão, como $2-x$, é transformada, primeiro, na soma $2+(-x)$ e só depois reescrita como $-x+2$, por comutação.

Essa forma de escrita é aliás a forma convencional e a mais desambiguada de se escrever muitas dessas expressões e aquela que, sem dúvida, a grande maioria das pessoas utilizaria. A forma mais desambiguada de se escrever a expressão

$2\log_e x$ seria precisamente essa, e não a expressão $(\log_e x)2$, onde seriam necessários parênteses para a desambiguar. Escrevê-la como $2\log_e x2$ seria, no mínimo, ambígua.

Com esse tipo de escrita é possível reduzir, de forma significativa, o número de pressupostos operacionais para cada classe. Por exemplo, no caso de somas ou subtracções, não é necessário verificar se os argumentos, à esquerda, são numéricos ou não. Caso o sejam, pode-se assumir que ambos o são.

A comutação de termos é também um método bastante eficiente de se reclassificar expressões. Por exemplo, uma expressão da classe *sum_diff_sum* pode ser reclassificada, por comutação, como uma expressão da classe *sum_sum_diff* . Com a reclassificação consegue-se reduzir, de forma significativa, a base axiomática para esses operadores.

A comutação de termos é também um tipo de reescrita contextualizada. A este tipo de transformação demos o nome de reescrita, por comutação, ou simplesmente reescrita-$\chi$ .

A reescrita-$\chi$ é apenas aplicável a estruturas $\Xi$-$op_1$, $\Xi$-$op_1$_$op_2$ e $\Xi$-$op_1$_$op_2$_$op_3$, onde $op_1 \in Sig_1^{\chi}$ e

$$Sig_1^{\chi} = \{sum, prod\}$$

$$C^{\chi} = \left\{ f(s,t) \in \mathcal{T} \mid f \in Sig_1^{\chi} \wedge s,t \in \mathcal{T} \right\}$$

**Definição 4.3-12** ***Regra de Reescrita-$\chi$***

> *Uma regra de reescrita-$\chi$ é uma regra de reescrita, condicional, representada por $\rightarrow_{\chi}$, que efectua transformações lógicas do tipo*

$s \to \left[\lambda_\chi(r)\right]\sigma$, *onde* $s \in \mathcal{C}^\chi$ *e* $\lambda_\chi(r)$ *representa uma reescrita-*$\chi$.

As regras de reescrita-$\chi$ são regras de reescrita, condicionais, contextualizadas, baseadas nos predicados e ***commute*** / 7. As regras de reescrita -$\chi$ estão organizadas também por classe de reescrita $c \in \mathcal{C}^\chi$.

A aplicação de uma regra de reescrita-$\chi$ pode ser representada simbolicamente por uma expressão do tipo

**apply_relation**$(L \to \mathrm{R}, State, Rule) \leftarrow$

$\mathrm{C}_1, \ldots, \mathrm{C}_n,$

**commute**$(L \to \mathrm{R}, State, Rule),$

$\lambda_\chi(R).$

**commute**$(L \to \mathrm{R}, State, Rule) \leftarrow$

$\mathrm{C}_1, \ldots, \mathrm{C}_n,$

$\lambda_\chi(R).$

Uma reescrita-$\chi$, $\lambda_\chi(r)$, é uma reescrita, por comutação de termos, que resulta da composição lógica de um ou mais construtores-$\lambda$.

**Exemplo 4.3-24** ***Reescrita*** $\chi$*-sum*

*A expressão* $2+x$*, com uma assinatura* sum*, é convertida por reescrita contextualizada,* $\chi$-sum*, aplicando-lhe uma regra do tipo*

**R1.** $\mathrm{sum}(s,t) \to \mathrm{sum}(t,s)$

***if*** $s \in \mathbb{N} \cup \mathcal{K} \wedge t \in \{x\}$

*A aplicação dessa regra daria origem à seguinte transformação lógica*

$$2+x \rightarrow_{\chi} x+2$$

A implementação Prolog de uma regra $\chi$-***sum*** contextualizada está listada na Secção A.1.6 do Apêndice A e a implementação Prolog de uma regra $\chi$-***sum***, com o mesmo nome, que lhe serve de contexto, listada na Secção A.1.2.

## 4.4 A Resolução Simbólica de Expressões

Uma expressão matemática pode ser resolvida simbolicamente aplicando-lhe um certo número de estratégias que visem essencialmente a sua normalização. Cada operador matemático possui um plano estratégico que poderíamos considerar como típico desse operador. A resolução simbólica de expressões depende essencialmente da estrutura triangular dessas expressões e do facto desta poder ser decomposta num certo número de estruturas triangulares básicas bem definidas.

O resultado de uma resolução simbólica é, como sabemos, uma sequência de transformações lógicas, que resultam da aplicação sistemática de uma, ou mais, dessas estratégias. A essa sequência de transformações lógicas demos o nome de solução simbólica, ou traço da resolução. Nesse tipo de solução, está bem patente o tipo de raciocínio que foi empregue na resolução. Por exemplo, no caso da expressão, uma solução possível seria a seguinte cadeia de transformações

$$((1+2)+3)+4 \downarrow^{1} \rightarrow_{\kappa} (3+3)+4 \downarrow^{1} \rightarrow_{\kappa} 6+4 \downarrow^{1} \rightarrow_{\kappa} 10$$

Como se pode observar, nessa solução está bem patente a forma como a expressão foi resolvida. Cada uma das transformações efectuadas é o resultado da

aplicação de uma estratégia. A maioria das expressões, incluídas nesta tese, necessitam de várias estratégias e podem apresentar várias soluções.

Vejamos, então, um caso simples de resolução

**Exemplo 4.4-1** ***Resolução simbólica de*** $\log_e 4^2$

*A expressão* $\log_e 4^2$*, com uma estrutura do tipo* $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$*, pode ser resolvida aplicando as seguintes estratégias*

$$\log_e 4^2 \underbrace{\downarrow^1 \rightarrow_\kappa}_{\substack{reescritas \\ \downarrow\text{-}log \\ \kappa\text{-}pwr}} \log_e 16 \underbrace{\rightarrow_\gamma}_{\substack{reescrita \\ \gamma\text{-}log}} \log_e 2^4 \underbrace{\rightarrow_\alpha}_{\substack{reescrita \\ \alpha\text{-}log\_pwr}} 4\log_e 2$$

***Passo 1:***

*Uma estrutura desse tipo sugere, como preferencial, a aplicação de uma estratégia de simplificação, à direita, por cálculo, com reescrita* $\downarrow \kappa\text{-}log\_pwr$ *, isto é, uma reescrita* $\kappa$*-pwr (redução por cálculo), seguida de uma reescrita* $\downarrow$*-log (simplificação) (Figura 4.4.1).*

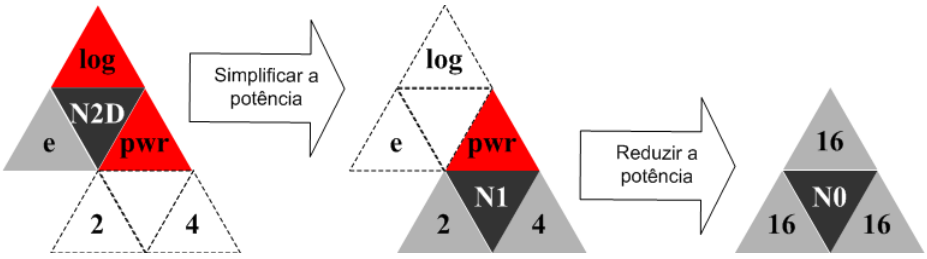



**Figura 4.4.1** ***Estratégia de simplificação à direita por cálculo***

***Passo 2:***

*A transformada,* $\log_e 16$, *é uma expressão com uma estrutura do tipo* $\Xi\text{-}log$, *o que sugere, como preferencial, a aplicação de uma estratégia de conversão axiomática, à direita, com reescrita contextualizada* $\gamma\text{-}log$, *(conversão por factorização) (Figura 4.4.2).*

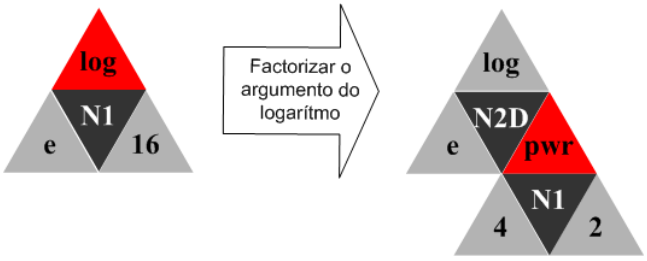



**Figura 4.4.2** ***Estratégia de transformação por factorização***

***Passo 3:***

*O resultado dessa aplicação,* $\log_e 2^4$, *é uma expressão com uma estrutura do tipo* $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$, *o que sugere também, como preferencial, a aplicação de uma estratégia de simplificação, à direita, por cálculo, com reescrita* $\downarrow\kappa\text{-}log\_pwr$ *(simplificação por cálculo). Só que essa estratégia é rejeitada pelo princípio de exclusão (Secção 4.4.2.2), pois a sua aplicação conduziria a uma expressão que já faria parte da memória colectiva (Secção 4.4.2.2) dessa resolução, ou seja, à expressão* $\log_e 16$. *É sugerida, então, por backtracking, que seja aplicada, como alternativa, uma estratégia de conversão com reescrita* $\alpha\text{-}log\_pwr$ *(conversão axiomática) à direita (Figura 4.4.3).*

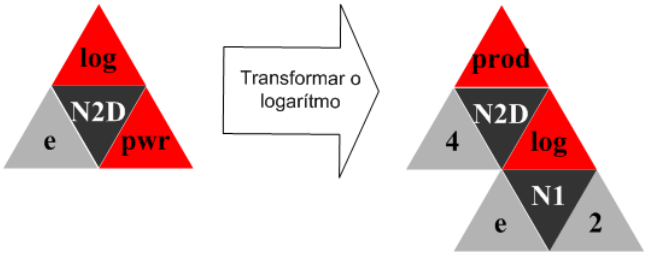



**Figura 4.4.3** *Estratégia de conversão*

*A expressão final,* $4\log_e 2$, *é uma expressão com uma estrutura do tipo* $[\Xi\text{-}prod\_log][\Xi\text{-}log]$, *irredutível. Trata-se, neste caso, da forma mais simples da expressão* $\log_e 4^2$.

---

Para garantir a unicidade da rescrita, qualquer tentativa de reescrita tem de passar pela memória colectiva da resolução. A unicidade da forma normal é assim garantida. Essa garantia é dada pelo princípio de exclusão.

Estratégias podem ser aplicadas a qualquer parte da estrutura e, portanto, é necessário saber em que posição da estrutura teve lugar a transformação. Isto, porque a reescrita de uma expressão envolve também o contexto em que está inserida.

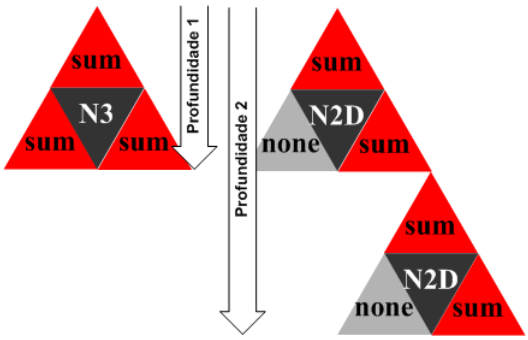



**Figura 4.4.4** *Profundidade de uma estrutura*

**Definição 4.4-1** ***Profundidade ou Nível Arbóreo***

> *A profundidade a que uma estrutura se encontra é definida pelo número de estruturas triangulares* Ξ*-op_op e* Ξ*-op_op_op , que se tem de atravessar para se chegar a ela. As estruturas* Ξ*-none e* Ξ*-op não afectam a profundidade a que uma estrutura se encontra.*

Por exemplo, uma expressão como $\log_e \sqrt{4^3}$ , com uma estrutura do tipo $[\Xi\text{-}log\_root][\Xi\text{-}root\_pwr][\Xi\text{-}pwr]$, possui uma profundidade de 2 ou dois níveis arbóreos[47]. Para se chegar à sua componente $\Xi\text{-}pwr$ , é necessário atravessar duas estruturas triangulares de nível 2, $\Xi\text{-}log\_root$ e $\Xi\text{-}root\_pwr$ , ou seja, aplicar duas estratégias de simplificação $-\downarrow$.

Podemos dizer que, numa estrutura triangular, a componente principal de uma expressão está à mesma profundidade que essa expressão. Assim, se uma expressão estiver à profundidade $n$ , a sua componente principal estará também a essa profundidade. O que coloca as restantes componentes a profundidades $n+k$ , com $k=1..r$ , onde $r$ é número restante de componentes. A profundidade a que uma componente se encontra é determinada, recursivamente, através da aplicação de estratégias de simplificação $-\downarrow$.

[47] É preciso não confundir o nível arbóreo de uma estrutura com o seu nível estrutural. Este último tem a ver essencialmente com nível estrutural de uma estrutura triangular básica. Por exemplo, uma estrutura triangular básica de nível 3, Ξ*-op_op_op* , pode estar a qualquer profundidade, numa estrutura arbórea.

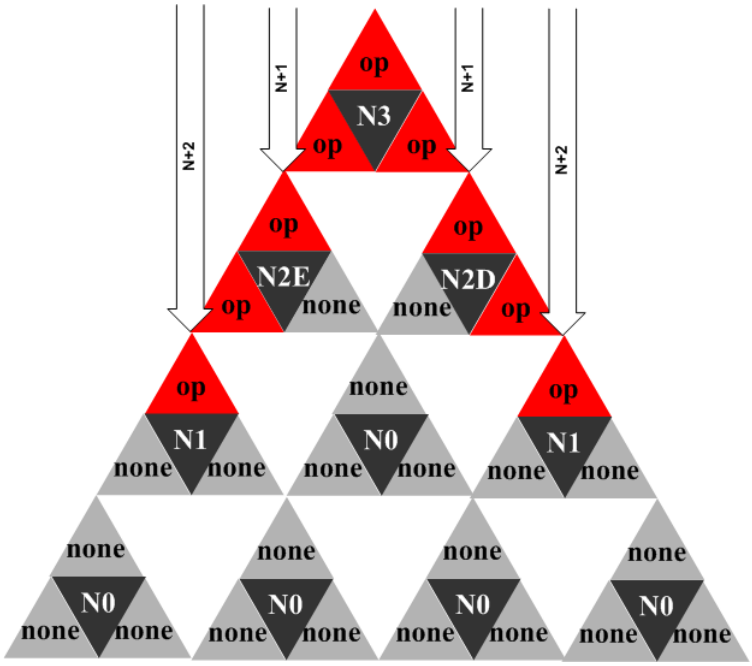



**Figura 4.4.5** *Estrutura arbórea*

Numa estrutura triangular, a posição $P$ de uma componente é determinada pela sua profundidade $Lv$, e a posição do vértice superior $P_0$ da sua estrutura triangular, ou seja,

$$P_i = 2P_0 + i \qquad (1)$$

onde $i = 1, 2$ representa o vértice inferior esquerdo e vértice inferior direito dessa estrutura. A posição do vértice superior $P_0$ de uma estrutura é determinada, recursivamente, através da fórmula (1). O vértice superior de uma estrutura triangular é 0 (estrutura de topo) ou um dos vértices inferiores da estrutura, à profundidade $Lv-1$, a que está ligada. A profundidade $Lv$ de uma estrutura arbórea é incrementada cada vez que é aplicada uma estratégia de simplificação à estrutura de topo.

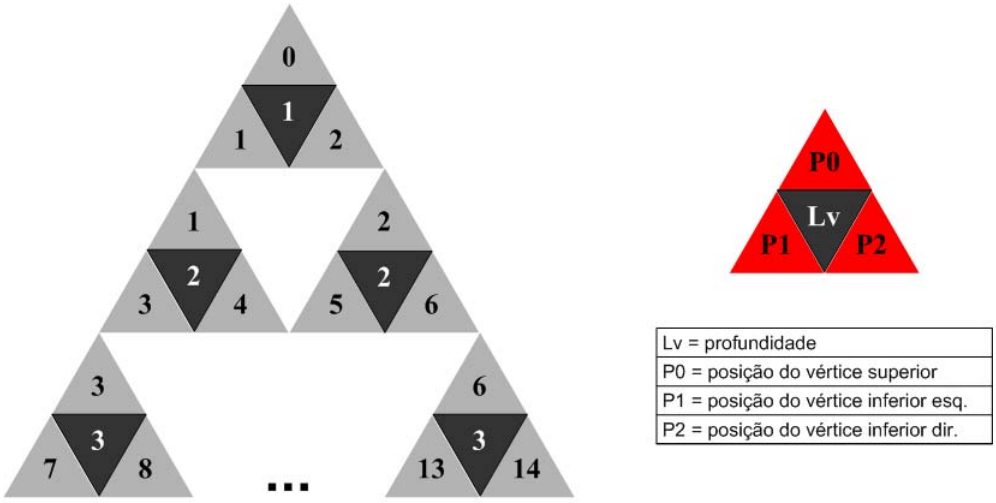



**Figura 4.4.6** ***Posições de uma estrutura*** $\Xi$

O processo é ilustrado na seguinte tabela

| *Lv* | $P_0$ | $P_1$ | $P_2$ | *# Estruturas* |
|---|---|---|---|---|
| 0 | $-1=\varepsilon$ | | | $2^0=1$ |
| 1 | 0 | $1=2(0)+1$ | $2=2(0)+2$ | $2^1=2$ |
| 2 | 1 | $3=2(1)+1$ | $4=2(1)+2$ | $2^2=4$ |
| 3 | 2 | $5=2(2)+1$ | $6=2(2)+2$ | $2^3=8$ |
| … | … | … | … | … |
| $n$ | $n-1$ | $P_i=2P_0+i$ | | $2^n$ |

***Legenda***:

*Lv* *Profundidade a que se encontra a estrutura*

$P_0$ *Posição do vértice superior*

$P_1$ *Posição do vértice inferior esquerdo*

$P_2$ *Posição do vértice inferior direito*

As estratégias são aplicadas a estruturas triangulares básicas, independentemente da posição que elas ocupam na estrutura triangular que as insere. As posições servem apenas para identificar essas estruturas no contexto arbóreo em que estão inseridas. As estruturas triangulares podem ser irredutíveis ($\mathcal{N}orm$), redutíveis ($\mathcal{R}edex$) ou potencialmente redutíveis ($\mathcal{R}edex$).

Podíamos então resumir que

- As estruturas triangulares de nível 0, $\Xi$-*none*, são sempre irredutíveis.
- As estruturas triangulares de nível 1, $\Xi$-*op*, são irredutíveis, ou redutíveis, por reescrita -$\kappa$ ou -$\tau$ (reduções $\Gamma_R$), ou potencialmente redutíveis, por reescrita -$\gamma$ (factorizações $\Gamma_T$).
- As estruturas triangulares de nível 2, $\Xi$-*op_op*, são irredutíveis, ou potencialmente redutíveis por reescrita -$\alpha$, -$\beta$ ou -$\psi$ (conversões $\Gamma_T$), ou por reescrita -$\downarrow$ (reduções $\Gamma_R$ ou conversões $\Gamma_T$).
- As estruturas triangulares de nível 3, $\Xi$-*op_op_op*, são irredutíveis, ou potencialmente redutíveis por reescrita -$\alpha$ (conversões $\Gamma_T$).

Quando tratadas como estruturas $[\Xi\text{-}op\_op]^2$, as estruturas de nível 3, $\Xi$-*op_op_op*, são equivalentes a duas estruturas de nível 2, ou seja, a uma estrutura arbórea de profundidade 2 (Figura 4.4.4).

### *4.4.1 As Estratégias de Resolução*

Numa resolução simbólica, podem ser aplicadas quatro (4) tipos de estratégias:

- Estratégias nulas ( E*n*C );
- Estratégias de redução ( E*n*R ou E*n*T );
- Estratégias de simplificação ( E*n*SE ou E*n*SD );
- Estratégias de conversão ( E*n*T )

A notação $EnX$ identifica o nível de estrutura $n$, a que a estratégia é aplicável e o tipo de reescrita $X$, que é efectuada.

Como vimos na Secção 4.3.1, existem quatro (4) tipos de reescrita:

- **Reduções**: Reescritas-$\kappa$ (identificada por $R$), ou reescritas-$\tau$ (identificada por $T$);
- **Simplificações**: Reescritas-$\downarrow$, à esquerda (identificada por $SE$) ou à direita (identificada por $SD$);
- **Conversões**: Reescritas-$\alpha$, -$\beta$, -$\gamma$, -$\psi$ e -$\chi$ (identificada por $T$);
- **Composições**: Reescritas-$\lambda$ (parte integrante de todas as outras reescritas).

A ausência de uma reescrita é identificada por $C$. De acordo com esta notação, uma estratégia E2SD seria interpretada como uma estratégia aplicável a estruturas de nível 2 ( $E2$ ), com reescrita por simplificação, à direita ( $SD$ ).

#### *4.4.1.1 A Estratégia Nula*

Computacionalmente, uma estratégia nula corresponde ao *insucesso* de todas as outras. Resolver, simbolicamente, uma expressão como 4 significa, simplesmente, deixá-la inalterada, pois a aplicação de qualquer outra, certamente, que falharia. Embora uma estratégia nula signifique, de facto, "*não fazer nada*", computacionalmente, significa "*tentar fazer algo, mas falhar*". As estratégias nulas

são apenas uma forma de confirmar que essas estruturas são, de facto, irredutíveis. Se nenhuma a reduz então o efeito seria nulo.

As estratégias nulas são designadas por estratégias E*n*C . Por exemplo, uma estratégia E0C seria uma estratégia nula ( *C* ) aplicável a uma estrutura de nível 0 , Ξ-*none* . A resolução de uma expressão como 4 seria o resultado da aplicação de uma estratégia desse tipo.

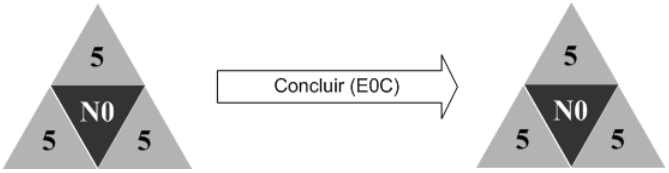



**Figura 4.4.7** ***Estratégia nula E0C***

Na presença de variáveis ou constantes, descontextualizadas, não se deve aplicar qualquer transformação lógica. Transformar $4$ em $2^2$ não faria qualquer sentido, pois essa expressão já se encontra representada na sua forma mais simples. Expressões como $x+1$ ou $\log_e x$, com estruturas do tipo Ξ-*op* , seriam consideradas também irredutíveis. A estratégia, a aplicar, neste caso, seria E1C .

### *4.4.1.2 A Estratégia de Redução*

As estratégias de redução são estratégias que visam, essencialmente, a redução de estruturas do tipo Ξ-*op* . Estratégias deste tipo utilizam apenas regras de reescrita-$\kappa$ (redução por cálculo) ou regras de reescrita-$\tau$ (redução por via axiomática). As estratégias de redução são estratégias de nível 1 que designamos por estratégias E1R ou E1T . Por exemplo, a expressão $4^2$ , com uma estrutura do tipo Ξ-*pwr* , é resolvida, simbolicamente, por uma estratégia de redução E1R [48], ou seja,

[48] Uma estratégia E1R é uma estratégia de redução, por cálculo, aplicada a estruturas de nível 1.

$$4^2 \underbrace{\rightarrow_{\kappa}}_{\text{E1R}} 16$$

Estratégias deste tipo reduzem, de facto, o grau de complexidade de uma estrutura.

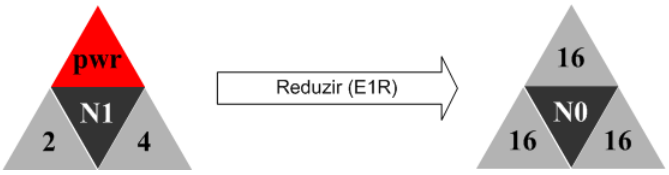



**Figura 4.4.8** *Estratégia de redução E1R*

Uma reescrita $\tau$-*sum* como, por exemplo, $x + x \rightarrow_{\tau} 2x$, é uma rescrita com elevado poder redutor. Uma reescrita deste tipo é o resultado da aplicação de uma estratégia de conversão, com um elevado poder redutor. Estratégia que designamos por E1T [49].

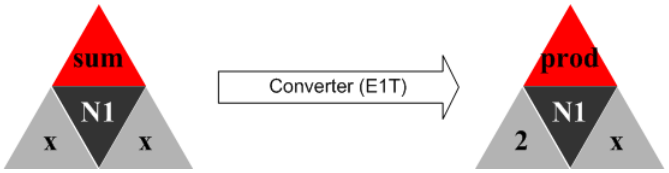



**Figura 4.4.9** *Estratégia de redução E1T redutora*

De acordo com a definição que foi dada na Secção 2.4.2, a expressão $2x$ é, de facto, a forma mais simples da expressão $x + x$.

### *4.4.1.3 A Estratégia de Simplificação*

As estratégias de simplificação são estratégias que visam, essencialmente, a redução ou conversão de componentes em estruturas do tipo Ξ-*op_op* ou

[49] Uma estratégia E1T é uma estratégia de redução, por via axiomática, aplicada a estruturas de nível 1.

$\Xi$-*op_op_op*. Note que uma estrutura $\Xi$-*op_op_op* é interpretada, neste caso, como uma estrutura do tipo $[\Xi\text{-}op\_op]^2$. Com este tipo de estratégias, são aplicadas reescrita-$\downarrow$ a seguir a outros tipos de reescrita. As estruturas alvo são componentes com estruturas redutíveis ou potencialmente redutíveis. As estratégias de simplificação são estratégias que designamos por estratégias E*n*SE (simplificações à esquerda) ou E*n*SD (simplificações à direita). Uma das estratégias que poderia ser aplicada à expressão $\log_e 4^2$, seria uma estratégia E2SD[50], com reescrita-$\kappa$. A expressão alvo, neste caso, seria a componente $4^2$, ou seja, uma expressão com uma estrutura do tipo $\Xi$-*pwr*. O resultado dessa aplicação seria uma reescrita $\downarrow\kappa$-*log_pwr*, ou seja, a aplicação de uma estratégia de simplificação à direita (E2SD), com redução por cálculo (E1R). A transformada seria a expressão $\log_e 16$, ou seja, uma expressão com uma estrutura triangular mais simples.

$$\log_e 4^2 \underbrace{\downarrow^1 \rightarrow_\kappa}_{\substack{\text{E2SD}\\ \text{E1R}}} \log_e 16 \rightarrow_\gamma \log_e 2^4 \rightarrow_\alpha 4\log_e 2$$

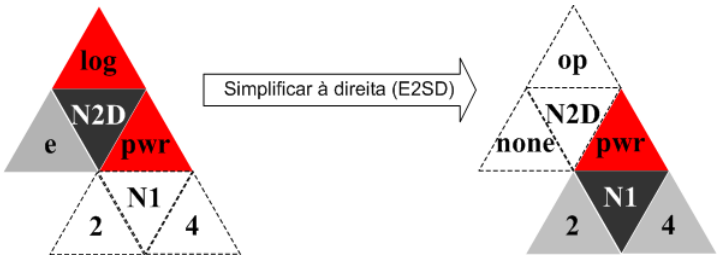



**Figura 4.4.10** *Estratégia de simplificação à direita E2SD*

### *4.4.1.4 A Estratégia de Conversão*

As estratégias de conversão são estratégias que visam, essencialmente, a conversão de estruturas do tipo $\Xi$-*op*, $\Xi$-*op_op* ou $\Xi$-*op_op_op*. As estratégias

[50] Uma estratégia E2SD é a uma estratégia de simplificação à direita ( ESD ) aplicada a estruturas de nível 2.

de conversão aplicam apenas regras de reescrita -$\alpha$ , -$\beta$ , -$\gamma$ , -$\psi$ ou -$\chi$ . A este tipo de estratégias demos o nome de estratégias E$n$T .

Para resolver expressões, como $\sqrt{8}$ , com uma estrutura do tipo $\Xi$-*root* , é necessário aplicar uma estratégia E1T , com reescrita contextualizada $\gamma$-*root* , ou seja, uma conversão axiomática, potencialmente redutora.

$$\sqrt{8} \underbrace{\rightarrow_{\gamma}}_{\text{E1T}} \sqrt{2^3} \rightarrow_{\alpha} 2^1\sqrt{2^1} \downarrow^1 \rightarrow_{\tau} 2\sqrt{2^1} \downarrow^2 \rightarrow_{\tau} 2\sqrt{2}$$

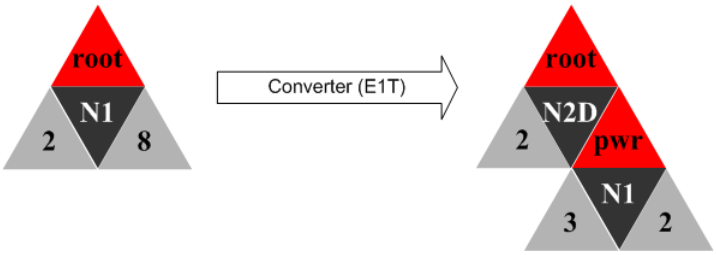



**Figura 4.4.11** ***Estratégia de conversão E1T potencialmente redutora***

No caso de uma expressão, como $\log_e 4^2$ , com uma estrutura do tipo $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$ , pode-se aplicar uma estratégia de conversão E2T , com reescrita $\alpha$-*log_pwr* , ou seja, uma conversão axiomática, bidireccional, não contextualizada e potencialmente redutora.

$$\log_e 4^2 \underbrace{\rightarrow_{\alpha}}_{\text{E2T}} 2\log_e 4 \downarrow^1 \rightarrow_{\gamma} 2\log_e 2^2 \downarrow^1 \rightarrow_{\alpha} 2(2\log_e 2) \rightarrow_{\alpha}$$

$$\rightarrow_{\alpha} 2(2)\log_e 2 \downarrow^1 \rightarrow_{\alpha} 4\log_e 2$$

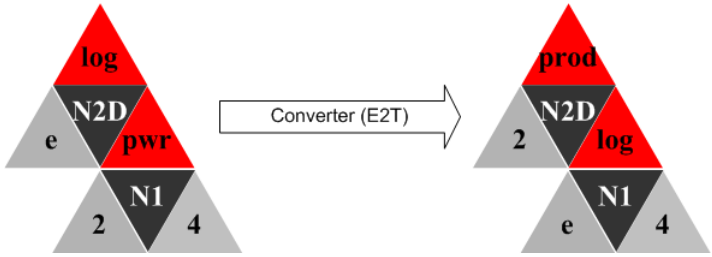



**Figura 4.4.12** *Estratégia de conversão E2T potencialmente redutora*

No caso de uma expressão, como $\frac{\mathrm{d}}{\mathrm{dx}}x^2$, com uma estrutura do tipo $[\Xi\text{-}der\_pwr][\Xi\text{-}pwr]$, pode-se aplicar uma estratégia de conversão E2T, com reescrita $\psi\text{-}der\_pwr$, ou seja, uma conversão axiomática, unidireccional, não contextualizada, redutora.

$$\frac{\mathrm{d}}{\mathrm{dx}}x^2 \underbrace{\rightarrow_{\psi}}_{\text{E2T}} 2x$$

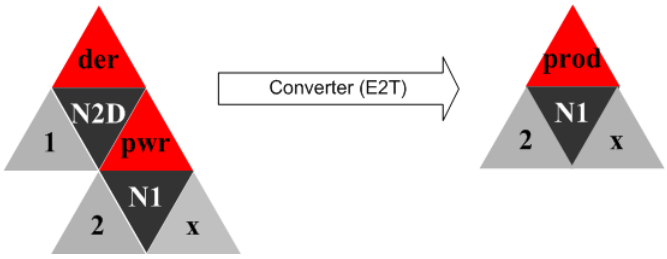



**Figura 4.4.13** *Estratégia de conversão E2T redutora*

No caso de uma expressão como $\log_e(2x)+\log_e(3x)$, com uma estrutura do tipo $[\Xi\text{-}sum\_log\_log][\Xi\text{-}log]^2$, pode-se aplicar uma estratégia de conversão E3T, com reescrita $\alpha\text{-}sum\_log\_log$, ou seja, uma conversão axiomática, bidireccional, não contextualizada e potencialmente redutora.

$$\log_e(2x)+\log_e(3x)\underbrace{\rightarrow_\alpha}_{\text{E3T}}\log_e((2x)(3x))\downarrow^1\rightarrow_\alpha \log_e((2(3))(x(x)))\downarrow^2\rightarrow_\kappa$$

$$\downarrow^2\rightarrow_\kappa \log_e(6(x(x)))\downarrow^2\rightarrow_\alpha \log_e(6(x^2))\rightarrow_\alpha \log_e 6+\log_e x^2\downarrow^1\rightarrow_\alpha$$

$$\downarrow^1\rightarrow_\alpha \log_e 6+2\log_e x$$

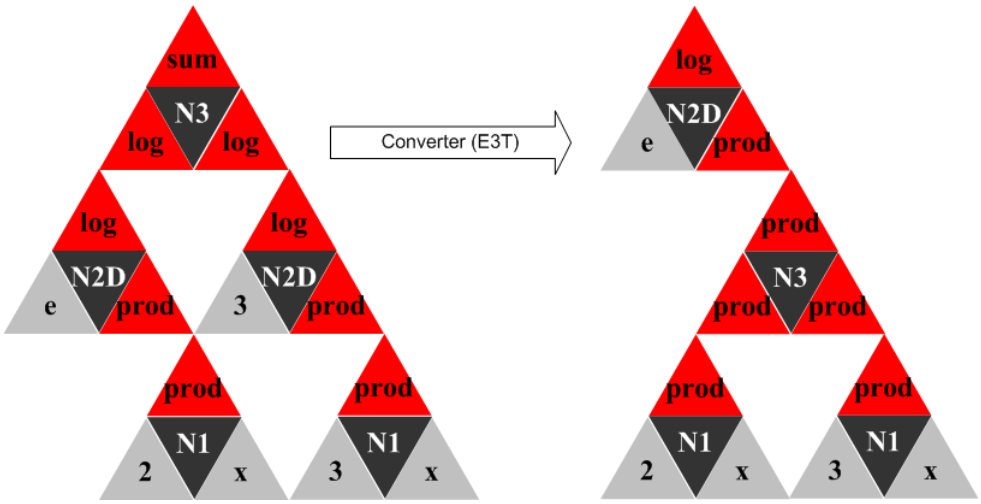



**Figura 4.4.14** *Estratégia de conversão E3T potencialmente redutora*

Neste exemplo foram aplicadas as seguintes estratégias:

| *Estratégias* | *Transformações* |
|---|---|
| E3T | $\log_e(2x)+\log_e(3x)\rightarrow_\alpha \log_e((2x)(3x))$ |
| E2SD | $\log_e((2x)(3x))\downarrow^1\rightarrow_\alpha \log_e((2(3))(x(x)))$ |
| E2SD , E2SE | $\log_e((2(3))(x(x)))\downarrow^2\rightarrow_\kappa \log_e(6(x(x)))$ |
| E2SD , E2SD | $\log_e(6(x(x)))\downarrow^2\rightarrow_\alpha \log_e(6(x^2))$ |
| E2T | $\log_e(6(x^2))\rightarrow_\alpha \log_e 6+\log_e x^2$ |

| | |
|---|---|
| E2SD | $\log_e 6 + \log_e x^2 \downarrow^1 \rightarrow_\alpha \log_e 6 + 2\log_e x$ |
| E3T | $\log_e 6 + \log_e x^2 \rightarrow_\alpha 2\log_e x + \log_e 6$ |

Basicamente, a escolha de estratégias resume-se a uma simples escolha entre dois tipos de estratégias – retardada ("*lazy*") ou estrita ("*strict*" ou "*eager*"). Por exemplo, no caso da expressão $\log_e 4^2$, a aplicação de uma estratégia de conversão $E2T$, com reescrita $\alpha$-*log_pwr*, seria considerada retardada, enquanto que a aplicação de uma estratégia de simplificação $E2SD$, com reescrita $\downarrow \kappa$-*log_pwr*, seria considerada estrita.

Em geral, deve-se escolher estratégias que reduzam, o mais rapidamente possível, o grau de complexidade da expressão. Optámos por estratégias estritas, como preferenciais, isto é, estratégias que incidam, de preferência, sobre os *redexes* mais interiores. Mas, com esse tipo de estratégias, nem sempre é possível garantir que as soluções obtidas sejam as mais curtas. Podemos, no entanto, garantir que todas elas são correctas.

### *4.4.2 A Reescrita Axiomática*

Como é fácil de depreender, uma expressão, como $1+2$, possui apenas uma solução[51] e a sua resolução envolve apenas um passo. Essa seria, de facto, a solução mais simples que alguém poderia, porventura, encontrar numa resolução simbólica. Mas, como é óbvio, nem todas as resoluções simbólicas possuem soluções assim tão simples, ou directas, como essa. Existem soluções simbólicas com dezenas, ou mesmo, centenas de passos. E, em muitos casos, soluções múltiplas.

O que realmente importa numa resolução é a leitura operacional que se faz da expressão. É a partir dessa leitura que se obtém o significado operacional da

[51] Duas se considerarmos a comutação das parcelas como uma possível solução.

expressão e que se põe em evidência a sua estrutura triangular. Com o conhecimento dessa estrutura, é possível identificar e classificar as suas várias componentes. O raciocínio que é aplicado, numa resolução simbólica, depende fortemente desse tipo de conhecimento. Por exemplo, a expressão $1+2+3+4+5$, possui várias soluções, todas elas diferentes, mas que conduzem ao mesmo resultado que, neste caso, é o valor numérico da expressão. A diferença entre elas está, precisamente, na forma como a expressão é lida. Uma possível solução seria a seguinte

$$((1+2)+3)+(4+5)\downarrow^{2}\rightarrow_{\kappa}(3+3)+(4+5)\downarrow^{1}\rightarrow_{\kappa}$$

$$\downarrow^{1}\rightarrow_{\kappa}6+(4+5)\downarrow^{1}\rightarrow_{\kappa}6+9\rightarrow_{\kappa}15$$

Mas essa não seria a única forma de se resolver essa expressão. Existem tantas quantas as leituras possíveis que pudemos fazer da sua estrutura. Nenhuma delas se poderia intitular como sendo a melhor. O que existem são apenas soluções correctas de leituras também correctas.

Note que numa resolução simbólica, como a que está ilustrada acima, são utilizadas, normalmente, várias estratégias de simplificação. Estratégias que classificamos como estritas, pois envolvem uma ou mais reescritas - $\downarrow$. Estratégias deste tipo têm por objectivo resolver, primeiro, os *redexes* mais interiores [DER01]. Note que os *redexes* vão sendo formados à medida que a resolução progride. A ordem em que os *redexes* são resolvidos não é importante, embora, por convenção, tenhamos optado por uma busca, em profundidade, da esquerda para a direita ("*depth-first*") . Essa é aliás a estratégia, normalmente, utilizada em cálculos numéricos, que envolvam expressões com parênteses. Segundo Valença e Barros [VAL00], as estratégias estritas são as preferidas, do ponto de vista computacional.

Uma leitura correcta deve pôr em destaque a estrutura da expressão. A

leitura operacional faz exactamente isso. Põe em destaque a sua estrutura triangular. Mas, no caso de haver mais do que uma, qual das leituras deveria ser considerada a *melhor* estratégia para resolver essa expressão? Nenhuma, pois o que interessa para esta tese não são soluções *melhores* ou *piores* mas sim soluções *correctas* de leituras, também, correctas.

Mas, como vamos ver, nem todas as expressões, estudadas para esta tese, possuem leituras múltiplas. Expressões, como $\log_e 4^2$, possuem apenas uma única leitura. Mas mesmo estas podem ser resolvidas de formas diferentes, através das leituras múltiplas que se podem fazer de algumas das suas componentes, ou das leituras múltiplas que se podem fazer de algumas das suas formas equivalentes. Por exemplo, a expressão $\log_e 4^2$ podia ser lida como uma expressão do tipo $[\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$. A estratégia sugerida, neste caso, como preferencial, seria uma estratégia estrita. Ou seja, uma estratégia com reescrita $\downarrow\kappa\text{-}log\_pwr$. Mas existem outras formas de a resolver. Todas elas empregam estratégias retardadas. Por exemplo, podia-se ter optado por resolver, primeiro, a componente com a estrutura $\Xi\text{-}log\_pwr$, ou seja, aplicar uma estratégia de conversão, E2T, com reescrita $\alpha\text{-}log\_pwr$. Estratégia que classificaríamos como retardada, pois deixaria um *redex*, mais interior, por resolver.

Todas estas formas alternativas de resolução diferem apenas na escolha da componente, ou seja, na ordem em que essas componentes devem ser lidas. A escolha de uma componente depende, também, da memória colectiva da resolução, ou seja, da existência, ou não, dessa componente como uma forma resolvida dessa expressão. Essa decisão é controlada pelo princípio de exclusão. Ou seja, a estratégia é automaticamente rejeitada se a reescrita conduzir a estados já visitados da resolução. Uma outra estratégia é então sugerida, como alternativa, por *backtracking*.

Em geral, as estratégias retardadas tendem a agravar o grau de complexidade das expressões. Mas como é óbvio, nem todas o fazem. Tudo depende dos *redexes* formados. Por exemplo, a transformação $\log_e 4^2 \rightarrow 2\log_e 4$ é retardada, mas potencialmente redutora. De facto, a expressão $2\log_e 4$ é mais simples mas ainda potencialmente redutora através da sua componente $\log_e 4$.

As reescritas-$\alpha$ são, como sabemos, redutoras ou potencialmente redutoras. Uma conversão, com reescrita-$\alpha$, reduz o que, normalmente, designamos por *redex* mais exterior. As reescritas-$\alpha$ ou -$\psi$, tanto pode ser estritas como retardadas. Tudo depende do contexto. Se não existirem *redexes*, mais interiores, a reescrita é considerada estrita. Caso contrário, é considerada retardada. Por exemplo, uma estratégia, com reescrita $\psi$-*der_log*, aplicada a uma expressão, como $\frac{\mathrm{d}}{\mathrm{dx}}\log_e x^2$, seria considerada retardada, pois deixaria o *redex* $\log_e x^2$ por resolver. A expressão teria uma estrutura do tipo $[\Xi\text{-}der\_log][\Xi\text{-}log\_pwr][\Xi\text{-}pwr]$. O resultado da reescrita seria uma expressão, como $\frac{1}{x^2}\frac{d}{dx}x^2$, com uma estrutura do tipo $[\Xi\text{-}prod\_div\_der][\Xi\text{-}div\_pwr][\Xi\text{-}der\_pwr][\Xi\text{-}pwr]^2$, mais complexa.

As estratégias retardadas são, em geral, estratégias que poderíamos classificar como *produtoras* de estruturas. Mas, como é óbvio, nem todas o são. As estratégias de conversão, com reescritas-$\beta$ ou -$\gamma$, são estritas, mas potenciadoras de novos *redexes*. Por exemplo, uma estratégia E2T, com reescrita-$\beta$, aplicada a uma como $\log_e \sqrt{2}$, é considerada estrita, mas potenciadora de um novo *redex* $\log_e 2^{\frac{1}{2}}$.

As estratégias estritas são, por outro lado, estratégias que tendem a reduzir ou, quanto muito, manter o grau de complexidade das expressões. As estratégias

estritas são, portanto, estratégias que poderíamos classificar como *consumidoras* de estruturas. Todas as estratégias de redução, com reescrita $-\kappa$ ou $-\tau$, são consideradas estritas. As estratégias de simplificação ou conversão, com reescrita $-\alpha$ ou $-\psi$, são consideradas estritas se não existirem *redexes*, mais interiores, ainda por resolver. Caso contrário, são consideradas retardadas. Por exemplo, uma estratégia de simplificação E2SD, com reescrita $\alpha$-*log_pwr*, aplicada a uma expressão como $\frac{\mathrm{d}}{\mathrm{dx}}\log_e x^2$, seria considerada estrita. O resultado seria a expressão, $\frac{\mathrm{d}}{\mathrm{dx}}(2\log_e x)$, com uma estrutura do tipo $[\Xi\text{-}der\_prod][\Xi\text{-}prod\_log][\Xi\text{-}log]$, mais simples. Todas as simplificações, com reescritas $-\downarrow\kappa$ e $-\downarrow\tau$, são consideradas estritas.

O importante numa resolução simbólica é que esse aumento do grau de complexidade se traduza na formação de um ou mais *redexes*. As estratégias nulas não estratégias que não consumem nem produzem *redexes*. A ausência de *redexes* ou a impossibilidade de se os formar, faz com que a resolução termine. A presença de *redexes* ou a possibilidade de os formar, faz com que a resolução prossiga. Numa resolução simbólica, deve haver mais consumidores do que produtores. Uma reescrita que consuma *redexes* é considerada redutora. Uma que produza, é considerada potencialmente redutora.

A resolução simbólica de expressões envolve, portanto, dois tipos de estratégias – estritas e retardadas. Essas estratégias, como sabemos, podem ser reduções, conversões ou simplificações. As reduções são sempre consumidoras, enquanto que as outras podem ser consumidoras ou produtoras. O importante é que, numa resolução simbólica, sejam aplicadas mais estratégias consumidoras do que produtoras. A ordem em que essas estratégias são aplicadas depende apenas da estrutura triangular das expressões, intervenientes no processo, e da memória colectiva da resolução. A memória colectiva de uma resolução é um tema que iremos abordar mais adiante, na Secção 4.4.2.2.

Podemos dizer que a resolução simbólica de expressões visa, essencialmente, a normalização dessas expressões, através da aplicação de um número finito de reduções, conversões e simplificações. Cada operador matemático possui o seu próprio plano estratégico. Esse plano difere essencialmente no tipo de conversões consideradas prioritárias. Uma estratégia para logaritmos não tem que ser, necessariamente, a mesma que uma estratégia para potências. As soluções, ou partes de soluções, que tenham sido já encontradas, podem determinar o tipo de estratégia que deve ser aplicado a seguir.

A Figura 4.4.15 mostra o tipo de estratégias que são aplicadas numa resolução simbólica e a ordem em que elas podem ser aplicadas. O gráfico ilustra apenas o ciclo de resolução de uma estrutura triangular. A resolução de várias estruturas é apenas uma repetição desse ciclo. Por exemplo, uma redução pode ser aplicada depois de uma simplificação, se essa redução for a estratégia aplicada no ciclo seguinte. Em cada ciclo, as estratégias de redução (reescritas $-\kappa$ e $-\tau$) têm precedência sobre qualquer uma das outras duas (conversões ou simplificações). A seguir, nessa escala de precedências, vêm as conversões que designamos por prioritárias, as simplificações e finalmente as conversões que designamos por não prioritárias. As conversões prioritárias são conversões que consideramos como tendo prioridade sobre as simplificações da mesma classe. As conversões prioritárias são, portanto, uma forma simples, mas eficaz, de dar prioridade a certas estratégias retardadas.

O outro gráfico mostra o relacionamento dessas estratégias com a estrutura triangular. Por exemplo, as estratégias de conversão E1T, com reescritas $-\gamma$, e as estratégias de redução E1R e E1T, com reescritas $-\kappa$ e $-\tau$, estão relacionadas com estruturas de nível 1, ou seja, estruturas do tipo $\Xi$-*op*.

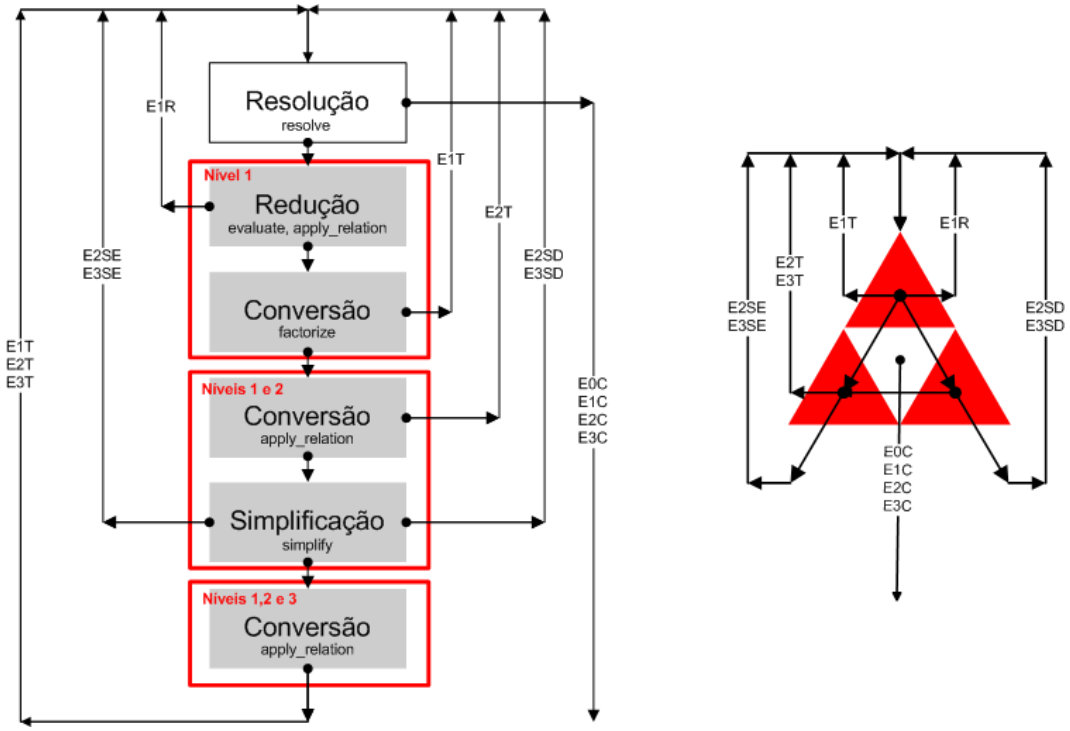



**Figura 4.4.15** *Estratégias de Resolução*

Como se pode ver pelo gráfico, numa resolução simbólica estão envolvidas várias estratégias, umas estritas e outras retardadas. Essas estratégias estão fortemente relacionadas com os vários tipos de reescrita que definimos na Secção 4.3.

Como está ilustrado na (Figura 4.4.15), podemos dizer que, em geral, numa resolução simbólica, a aplicação de estratégias envolve os seguintes passos e nessa ordem:

- A aplicação de estratégias de redução, com reescrita $-\kappa$ e $-\tau$ (E1R e E1T);
- A aplicação de estratégias de conversão, com reescrita $-\alpha$ e $-\gamma$ (E1T e E2T), que sejam consideradas prioritárias para essa classe de operadores;

- Estratégias de simplificação, com reescrita-$\downarrow$, à esquerda (E2SE e E3SE);
- Estratégias de simplificação, com reescrita-$\downarrow$, à direita (E2SD e E3SD);
- A aplicação de estratégias de conversão, com reescrita-$\alpha$, -$\beta$, -$\gamma$, -$\chi$ e -$\psi$ (E1T, E2T e E3T), que não sejam consideradas prioritárias para essa classe de operadores;
- A aplicação de estratégias nulas (E$n$C com $n = 0,1,2,3$).

A reescrita-$\lambda$ é comum a todas elas.

Vejamos, então, como é que seriam aplicadas essas estratégias no caso de uma expressão, como $\log_e 4^2$. As possíveis soluções estão apresentadas a seguir.

| *Solução* | *Passos* | *Ref$^n$* |
|---|---|---|
| **Solução** 1 | $\log_e 4^2 \rightarrow_\alpha 2\log_e 4 \downarrow^2 \rightarrow_\gamma 2\log_e 2^2 \downarrow^1 \rightarrow_\alpha 2(2\log_e 2) \rightarrow_\alpha$ $\rightarrow_\alpha 2(2)\log_e 2 \downarrow^1 \rightarrow_\kappa 4\log_e 2$ | (1) |
| **Solução** 2 | $\log_e 4^2 \downarrow^1 \rightarrow_\kappa \log_e 16 \downarrow^1 \rightarrow_\gamma \log_e 2^4 \rightarrow_\alpha 4\log_e 2$ | (2) |
| **Solução** 3 | $\log_e 4^2 \downarrow^2 \rightarrow_\gamma \log_e (2^2)^2 \downarrow^1 \rightarrow_\alpha \log_e 2^{2(2)} \downarrow^2 \rightarrow_\kappa$ $\downarrow^2 \rightarrow_\kappa \log_e 2^4 \rightarrow_\alpha 4\log_e 2$ | (3) |

Como se pode ver, em todas elas, são utilizadas estratégias estritas e retardadas. Na primeira solução (1), foi aplicada uma estratégia retardada, $\log_e 4^2 \rightarrow_\alpha 2\log_e 4$. Na segunda solução (2), foi aplicada uma estratégia estrita,

$\log_e 4^2 \downarrow^1 \rightarrow_\kappa \log_e 16$). Na terceira solução (3), foi de novo aplicada uma estratégia retardada, $\log_e 4^2 \downarrow^2 \rightarrow_\gamma \log_e (2^2)^2$. A aplicação dessas estratégias é feita por *backtracking*, sob o controlo do princípio de exclusão (Secção 4.4.2.2). Nesta altura, a aplicação de qualquer outra estratégia, estrita ou retardada, conduziria a estados já visitados, $\log_e 16$ ou $2\log_e 4$. O princípio de exclusão garante a unicidade de todas as formas equivalentes geradas pelo método de resolução simbólica.

### *4.4.2.1 O Processo de Reescrita*

Uma reescrita axiomática nem sempre conduz a formas simplificadas dessas expressões. Como vimos atrás, a aplicação de estratégias retardadas pode conduzir a expressões equivalentes mais complexas. Porém, a grande maioria das reescritas visa, essencialmente, a redução ou formação de *redexes*. As outras são, simplesmente terminais e reescrevem, portanto, formas normais dessas expressões. Por exemplo, numa reescrita contextualizada, como $2\log_e 4 \downarrow^1 \rightarrow_\gamma 2\log_e 2^2$, o objectivo é, precisamente, a formação do *redex* $\log_e 2^2$. A sua presença faz com que a expressão $\log_e 2^2$ seja redutível por uma reescrita $\downarrow \alpha$-*log_pwr* , seguida de uma reescrita $\downarrow \kappa$-*prod* .

O próprio cálculo numérico é tratado, nesta tese, de uma forma estritamente simbólica, por reescrita axiomática. Por exemplo, uma expressão numérica como $3+(2^3-(5+\sqrt{4}))$ é resolvida aplicando reduções, conversões e simplificações. Uma possível solução seria a seguinte cadeia de transformações

| ***Passos*** | ***Transformações*** | ***Ref[a]*** |
|---|---|---|
| ***1, 2*** | $3+(2^3-(5+\sqrt{4}))\downarrow^2 \rightarrow_\kappa 3+(8-(5+\sqrt{4}))\downarrow^3 \rightarrow_\gamma$ | (1) |
| ***3, 4*** | $\downarrow^3 \rightarrow_\gamma 3+(8-(5+\sqrt{2^2}))\downarrow^3 \rightarrow_\alpha 3+(8-(5+2))\downarrow^2 \rightarrow_\kappa$ | (2) |

| | | |
|---|---|---|
| ***5, 6, 7*** | $\downarrow^{2}\rightarrow_{\kappa} 3+(8-7)\downarrow^{1}\rightarrow_{\kappa} 3+1\rightarrow_{\kappa} 4$ | (3) |

O mesmo se poderia dizer de expressões como $2\dfrac{3+\dfrac{1}{4}}{6}$. Uma possível solução seria a seguinte cadeia de transformações

| ***Passos*** | ***Transformações*** | ***Ref[a]*** |
|---|---|---|
| ***1, 2, 3*** | $2\dfrac{3+\dfrac{1}{4}}{6}\downarrow^{2}\rightarrow_{\chi} 2\dfrac{\dfrac{1}{4}+3}{6}\downarrow^{2}\rightarrow_{\alpha} 2\dfrac{\dfrac{1+12}{4}}{6}\downarrow^{3}\rightarrow_{\kappa}$ | (1) |
| ***4, 5, 6, 7*** | $\downarrow^{3}\rightarrow_{\kappa} 2\dfrac{\dfrac{13}{4}}{6}\downarrow^{1}\rightarrow_{\alpha} 2\dfrac{13}{24}\rightarrow_{\alpha} \dfrac{26}{24}\rightarrow_{\gamma} \dfrac{13}{3}\dfrac{2}{2^{3}}\downarrow^{1}\rightarrow_{\gamma}$ | (2) |
| ***8, 9, 10*** | $\downarrow^{1}\rightarrow_{\gamma} \dfrac{13}{3}2^{1-3}\downarrow^{2}\rightarrow_{\kappa} \dfrac{13}{3}2^{-2}\downarrow^{1}\rightarrow_{\alpha} \dfrac{13}{3}\dfrac{1}{2^{2}}\downarrow^{2}\rightarrow_{\kappa}$ | (3) |
| ***11, 12*** | $\downarrow^{2}\rightarrow_{\kappa} \dfrac{13}{3}\dfrac{1}{4}\rightarrow_{\alpha} \dfrac{13}{12}$ | (4) |

A reescrita axiomática envolve a reescrita de expressões equivalentes. A unicidade dessas formas é garantida pelo princípio de exclusão. A memória colectiva de uma resolução garante a convergência dessas formas, pois é formada por todas as formas equivalentes dessa expressão, incluindo a sua forma mais simples.

Consideramos, portanto, que

> **o resultado de uma resolução simbólica é a forma mais simples dessa expressão se a base axiomática for completa.**

A forma mais simples de uma expressão não tem de ser necessariamente numérica. De facto, na sua grande maioria, essas formas são meramente simbólicas.

Segundo [VAL00], apenas as relações de igualdade semântica, entre expressões, sintacticamente, diferentes, podem contribuir para a normalização dessas expressões. No âmbito de uma resolução simbólica, só devem ser consideradas relações de equivalência que envolvam relações de igualdade semântica entre expressões, sintacticamente, diferentes.

Mas, como sabemos, nem todas as relações de equivalência são redutoras ou potencialmente redutoras. Por exemplo, relações como $5 \to 2+3$ não contribuem, certamente para a normalização de constantes. Como já sabemos, as constantes numéricas são expressões, por definição, consideradas irredutíveis. Mas, também, certas constantes simbólicas, como $\log_{10} 3$, são consideradas irredutíveis. Outras, porém, como por exemplo $\log_5 5$, são redutíveis, por reescrita axiomática.

Em princípio, uma expressão que não seja redutível, deve ser considerada irredutível. Uma transformação, por reescrita axiomática, é considerada potencialmente redutora se a sua transformada for redutível, mas mais complexa. Por exemplo, a transformação $2\log_e 4 \downarrow^1 \to_\gamma 2\log_e 2^2$ é potencialmente redutora.

Uma transformação, por reescrita axiomática, é considerada redutora, se a sua transformada for irredutível, ou redutível, mas mais simples. Por exemplo, as transformações $\log_e 2^4 \downarrow^1 \to_\alpha 4\log_e 2$ e $\log_e 4^2 \downarrow^1 \to_\alpha 2\log_e 4$ são ambas redutoras, mas a última é ainda redutível.

Assim, consideramos que

> **numa resolução simbólica só devem ser efectuadas transformações que sejam redutoras ou, potencialmente, redutoras.**

De facto, só faz sentido transformar expressões se o resultado for uma expressão mais simples, ou uma expressão mais complexa, mas que possa vir a tornar-se mais simples através da formação de novos *redexes*. Por exemplo, a regra

de derivação da soma é uma regra potencialmente redutora. Com a sua aplicação, são formados novos *redexes*.

Numa resolução simbólica, só se pode reescrever uma expressão, uma única vez. A reescrita só pode ter lugar se o resultado da transformação não interceptar a memória colectiva. Por exemplo, uma transformação, por reescrita axiomática, como $\log_e 4^2 \downarrow^1 \rightarrow_\kappa \log_e 16$, não teria lugar, se a expressão $\log_e 16$ já fizesse parte da memória colectiva dessa resolução. Qualquer reescrita, por reflexividade, simetria ou transitividade, é automaticamente rejeitada. Cada um desses casos está ilustrado a seguir.

| | ***Transformações*** | ***Ref[a]*** |
|---|---|---|
| **Reflexividade** | $\log_e 4^2 \not\rightarrow \log_e 4^2$ | (1) |
| **Simetria** | $\log_e 4^2 \rightarrow_\alpha 2\log_e 4 \not\rightarrow \log_e 4^2$ | (2) |
| **Transitividade** | $\log_e 4^2 \rightarrow_\alpha 2\log_e 4 \downarrow^1 \rightarrow_\gamma 2\log 2^2 \rightarrow_\beta$ | (3) |

Uma transformação, por reflexividade (1), corresponderia, de facto, à aplicação de uma regra de reescrita do tipo $l \rightarrow r$ onde $l \equiv r$, ou seja, à aplicação de uma relação onde, na realidade, não seria efectuada nenhuma transformação. Regras deste tipo não são nem redutoras nem potencialmente redutoras e portanto não fazem parte da base axiomática.

Uma transformação por simetria (2) corresponde à aplicação de uma regra de reescrita do tipo, $l \rightarrow r$, em ambos os sentidos. Sabe-se [DER01] que transformações deste tipo podem produzir cadeias de derivação infinita, ou seja, sequências do tipo $s \rightarrow t \rightarrow s \rightarrow t \rightarrow \cdots$. Regras deste tipo são redutoras ou potencialmente redutoras, mas controladas pelo princípio de exclusão.

Uma transformação, por transitividade (3), corresponde à aplicação de uma

regra de reescrita do tipo $l \to r$, cujo resultado $r$, já tinha sido obtido anteriormente através de uma outra regra. Regras deste tipo são também redutoras ou potencialmente redutoras e igualmente controladas pelo princípio de exclusão.

### *4.4.2.2 A Memória Colectiva e o Princípio de Exclusão*

A memória colectiva de uma resolução é apenas um mapa de equivalências obtida, por reescrita axiomática, para essa resolução. O conjunto de todas as formas equivalentes de uma expressão, obtidas por resolução simbólica, $\mathcal{R}es(s)$, é um subconjunto próprio da sua classe de equivalência $\mathcal{E}q(s)$, ou seja, $\mathcal{R}es(s) \subset \mathcal{E}qu(s)$. A memória colectiva de uma resolução $\mathcal{MR}es(s)$ é, portanto, o conjunto de todas as transformações lógicas efectuadas sobre a expressão $s \in \mathcal{T}$, por reescrita axiomática. Se identificarmos os elementos dessa memória por $id(i, j)$, onde $i$ e $j$ representam respectivamente a tentativa de resolução e o respectivo passo nessa tentativa, então os elementos da memória podem ser representados por $step(id(i, j), \Gamma_j(s))$, onde $\Gamma_j(s)$ representa a $j$-gésima transformação lógica efectuada sobre $s$. Os elementos da memória estão ordenados, lexicograficamente, por $id(i, j)$, ou seja,

$$step(id(1,1), \Gamma_1(s)) >_\Gamma step(id(1,2), \Gamma_2(s)) >_\Gamma \cdots >_\Gamma step(id(i, j), \Gamma_j(s)) >_\Gamma \cdots$$

Os elementos da memória são instâncias do predicado ***step*** / 11.

**Definição 4.4-2** ***Memória Colectiva***

> *A memória colectiva de uma resolução, $\mathcal{MR}es(s)$, é o conjunto ordenado de todas as transformações lógicas, $\Gamma_j(s)$, efectuadas sobre a expressão $s \in \mathcal{T}$ definida como objecto dessa resolução.*

A instanciação da memória colectiva é baseada no predicado ***assert_step*** / 11. A implementação Prolog desse predicado está listada na Secção

A.3.2 do Apêndice A.

Para compreendermos melhor o papel que é desempenhado pela memória colectiva, numa resolução simbólica, examinemos a resolução de uma expressão como $\log_e 4^2$. As três possíveis soluções[52] estão ilustradas a seguir.

| ***Solução*** | ***Passos*** | ***Ref[a]*** |
|---|---|---|
| **Solução** 1 | $\log_e 4^2 \rightarrow_\alpha 2\log_e 4 \downarrow^2 \rightarrow_\gamma 2\log_e 2^2 \downarrow^1 \rightarrow_\alpha 2(2\log_e 2) \rightarrow_\alpha$ $\rightarrow_\alpha 2(2)\log_e 2 \downarrow^1 \rightarrow_\kappa 4\log_e 2$ | (1) |
| **Solução** 2 | $\log_e 4^2 \downarrow^1 \rightarrow_\kappa \log_e 16 \downarrow^1 \rightarrow_\gamma \log_e 2^4 \rightarrow_\alpha 4\log_e 2$ | (2) |
| **Solução** 3 | $\log_e 4^2 \downarrow^2 \rightarrow_\gamma \log_e (2^2)^2 \downarrow^1 \rightarrow_\alpha \log_e 2^{2(2)} \downarrow^2 \rightarrow_\kappa$ $\downarrow^2 \rightarrow_\kappa \log_e 2^4 \rightarrow_\alpha 4\log_e 2$ | (3) |

Todas elas conduzem a expressão $\log_e 4^2$ à sua forma mais simples, $4\log_e 2$, passando por um certo número de passos ou estados intermédios. Na primeira (1), a expressão $2\log_e 4$ é obtida por aplicação de uma estratégia retardada $\alpha$-*log_pwr*. Na segunda (2), a expressão $\log_e 16$ é obtida por aplicação de uma estratégia estrita $\downarrow \kappa$-*pwr* Na terceira (3), a expressão $\log_e (2^2)^2$ é obtida por aplicação de uma estratégia retardada $\downarrow \gamma$-*pwr*. As duas últimas estratégias foram obtidas por *backtracking* mas induzido pelo princípio de exclusão.

De acordo com este princípio, uma transformação lógica só pode ter lugar, se o resultado da sua reescrita não fizer parte da memória colectiva dessa resolução. Caso contrário, a regra que foi aplicada é excluída e uma outra seleccionada por *backtracking*. Note que a decisão envolve qualquer transformação com a mesma

[52] O símbolo - $\downarrow$ para indicar que a substituição teve lugar a um nível mais interior da estrutura.

reescrita. Podemos, portanto, dizer que qualquer tentativa de resolução que intercepte a memória, é automaticamente rejeitada pelo princípio de exclusão.

No caso de não existirem estratégias alternativas, a solução é completada a partir da memória colectiva. Por exemplo, a última solução (3) foi completada dessa forma. De facto, a transformação $\log_e 2^4 \rightarrow_\alpha 4\log_e 2$ foi obtida a partir da memória, uma vez que a expressão $\log_e 2^4$ já fazia parte da *memória colectiva* dessa resolução.

Note que todas essas expressões são formas equivalentes da expressão $\log_e 4^2$, e as únicas que podem ser obtidas, por reescrita axiomática. Todas elas fazem parte da memória colectiva dessa resolução. Por exemplo, a expressão $\log_e 16$ pode ser obtida, por reescrita axiomática, a partir de expressões como $\log_e 4^2$ e $\log_e 2^4$, desde que a reescrita não viole o princípio de exclusão. Segundo Dershowitz [DER01], as expressões $\log_e 4^2$ e $\log_e 2^4$ são uníveis, pois ambas convergem, por rescrita, para a mesma expressão, neste caso, $\log_e 16$. Uma vez reescrita, a expressão só pode ser obtida a partir da memória. O que significa que na memória só pode existir uma cópia dessa expressão. Naturalmente, isso implica uma possível redução do número de soluções que, teoricamente, poderiam ser obtidas, por reescrita, pois esse facto impede que uma solução possa intersectar uma outra mais do que uma vez, ou que possa interceptar mais do que uma solução na memória.

A terminação de uma resolução é garantida pelo princípio de exclusão e pela finitude da base axiomática. De facto, o princípio de exclusão garante a unicidade de qualquer expressão obtida por reescrita axiomática. A finitude da base axiomática garante que só um número finito de regras pode ser aplicado. As formas equivalentes de uma expressão só podem interceptar a sua memória, uma única vez. Isto é, na memória não podem existir formas duplicadas dessas expressões. A

memória colectiva de uma resolução é, por definição, o conjunto ordenado de todas as reescritas axiomáticas efectuadas sobre a expressão que é objecto dessa resolução (Secção 4.4.2). Isso inclui, naturalmente, a forma mais simples dessa expressão. De facto, o fecho transitivo de uma reescrita axiomática é sempre a forma mais simples da expressão. Isso é garantido pelo facto da base axiomática conter apenas regras redutoras ou potencialmente redutoras.

A confluência de uma resolução simbólica é garantida pela memória colectiva dessa resolução e pela completude da sua base axiomática. Pelo princípio de exclusão, uma solução só pode interceptar uma outra, uma única vez. Porém, qualquer solução pode ser interceptada mais do que uma vez. Isto é, a intercepção entre duas soluções, quaisquer, é um conjunto singular ("*singleton*"). A forma interceptada é, precisamente, o resultado da reescrita para a qual não existem estratégias alternativas para a sua resolução. Uma vez reescrita, a expressão só pode ser obtida a partir da sua memória. O que significa que uma solução que intersecte uma outra pode ser completada a partir da sua memória. Como, por definição, uma solução é um conjunto ordenado de transformações lógicas obtidas por reescrita axiomática, todas as soluções possíveis de uma resolução simbólica devem fazer parte da sua memória colectiva. Se a base axiomática for completa, então qualquer solução deve conduzir à forma mais simples da expressão. De facto, a completude de uma base axiomática garante a irredutibilidade de uma reescrita axiomática. Assim, se existir mais do que uma solução, as soluções devem interceptar uma outra, pelo menos, uma vez. De facto, se tal não acontecesse, existiriam na memória soluções que não conduziriam à forma mais simples, o que seria uma contradição.

Podemos, portanto, dizer que todas as soluções obtidas por resolução simbólica convergem para a forma mais simples da expressão, se a base axiomática for completa. De facto, a forma mais simples é aquela para a qual todas as intercepções convergem. Podemos também dizer que, se a base axiomática for completa, o resultado de uma resolução não depende da estratégia utilizada.

Podemos, portanto, concluir que, se a base axiomática for completa, a resolução simbólica de expressões por reescrita axiomática é convergente.

Na Figura 4.4.16 estão ilustradas várias soluções e as possíveis intersecções que podem ocorrer entre elas. Nessa figura, as expressões estão identificadas pela ordem em que foram reescritas. A expressão a resolver é a expressão identificada por $0$. A expressão $7$ é a forma mais simples. As soluções que terminam nas expressões $10$, $6$ e $16$, são completadas a partir da memória e, portanto, convergem também para a mesma expressão. Note que uma solução só pode interceptar uma outra, mas pode ser interceptada por mais do que uma. Podemos, então, dizer que a base axiomática é completa, se todas as soluções interceptarem, pelo menos uma outra solução na memória. Em todos os testes que efectuámos, esse facto foi amplamente verificado.

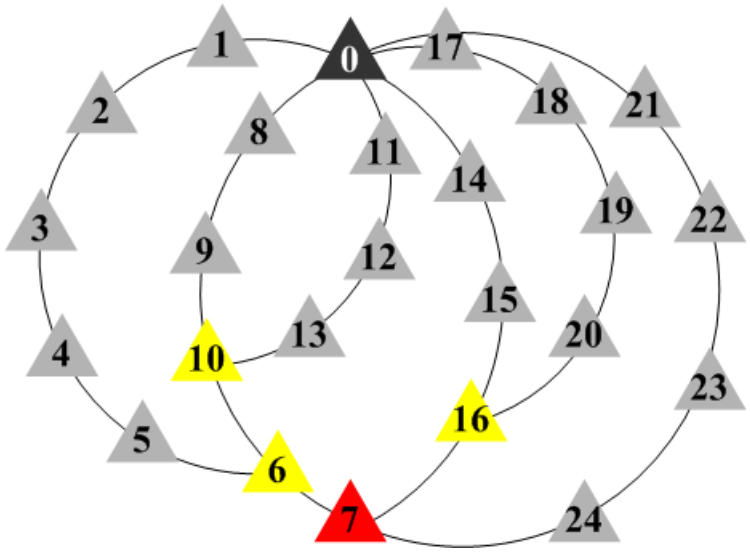



**Figura 4.4.16** *Memória Colectiva*

O método de resolução que desenvolvemos é baseado nos predicados ***resolve***$/[2,6]$. A implementação Prolog do predicado ***resolve***$/6$ está listada na Secção A.2.2 do Apêndice A.

A escolha de uma estratégia depende do operador principal da expressão que se pretende resolver. Cada operador possui o seu próprio plano estratégico. Esse plano, na sua forma geral, é comum a todos os operadores. Por exemplo, para se simplificar uma expressão como $\log_e \sqrt{4^3}$, com uma estrutura do tipo $\Xi$-*log_root*, deve-se aplicar uma estratégia, certamente, diferente da que seria, porventura, escolhida para resolver uma expressão, como $\log_e(\sqrt{4})^3$, com uma estrutura do tipo $\Xi$-*log_pwr*.

A implementação Prolog de uma estratégia $\gamma$-***log_root*** está listada na Secção A.1.2 do Apêndice A.

A aplicação das regras de reescrita é controlada pelo princípio de exclusão.

**Definição 4.4-3** ***Princípio de Exclusão***

> *Seja $\rightarrow_\Gamma$ uma regra de reescrita, que efectua uma transformação lógica $\Gamma$. De acordo com o princípio de exclusão, a regra só pode ser aplicada se o resultado dessa reescrita não fizer parte da memória colectiva da resolução.*

A consulta à memória colectiva de uma resolução é baseada no predicado ***except_in***/9. A implementação Prolog desse predicado está listada na Secção A.2.2 Apêndice A.

A memória colectiva de uma resolução é construída durante a resolução simbólica de uma expressão e pode envolver uma, ou mais, sessões práticas. Cada sessão prática resolve a expressão, tendo em conta a sua memória colectiva.

Como vimos atrás, a expressão $\log_e 4^2$ possui várias soluções simbólicas. Uma solução simbólica é apenas uma sequência lógica de formas equivalentes obtidas por aplicação de uma, ou mais, estratégias de resolução. Para se obterem

todas as soluções é necessário recorrer à memória colectiva da resolução. A memória colectiva funciona como um agente verificador de reescritas. É, precisamente, através da memória colectiva que o processo de *backtracking* é controlado. O que resulta, naturalmente, numa procura exaustiva de outras formas equivalentes dessa expressão. Através deste método não é possível obter todas as formas equivalentes de uma expressão. Apenas aquelas que podem ser obtidas por reescrita axiomática e para as quais existem regras adequadas na base axiomática.

O tamanho da memória depende apenas do número de soluções possíveis, do número de passos de cada solução, e do tamanho das expressões obtidas. Por exemplo, a memória colectiva de $\log_e 4^2$ é constituída apenas pelas formas listadas na tabela e as respectivas transformações lógicas.

A resolução simbólica de uma expressão conduz sempre a uma forma normal (irredutível) dessa expressão. Essa forma é única se a base axiomática for convergente para todas as classes envolvidas nessa resolução. Esse facto foi verificado, empiricamente, para um grande número de expressões. Todas as formas obtidas, por reescrita axiomática, são estruturas redutoras, potencialmente redutoras ou normais. Todas as soluções que intersectem a memória colectiva da resolução, conduzem à mesma forma normal. Soluções paralelas podem, ou não, conduzir à mesma forma normal. A questão da unicidade da forma normal é indecidível, pois depende, única e exclusivamente, da completude da base axiomática.

Em todos os testes que conduzimos, o resultado foi sempre a forma mais simples da expressão, ou seja, a sua única forma normal, o que nos levou a concluir que, para essas expressões, a base axiomática estava completa, e que as soluções obtidas dependiam apenas das estratégias aplicadas. Como vimos, as estratégias afectam apenas a forma de chegar a esse resultado. Uma solução simbólica representa apenas uma forma diferente de resolver, simbolicamente, uma expressão matemática.

Vimos também que o resultado de uma resolução simbólica não tem que ser, necessariamente, numérico. De facto, a grande maioria dos resultados obtidos são, meramente, simbólicos. Por exemplo, a forma mais simples da expressão $\log_e 4^2$ é a expressão $4\log_e 2$. Esse facto não é, porém, de todo evidente. Só se torna evidente através de uma resolução simbólica.

Resolver uma expressão numérica, de forma simbólica, não é obter apenas o valor numérico que essa expressão possa, porventura, representar. É mais do que isso. Resolver essa expressão simbolicamente é obter uma ou mais soluções simbólicas que conduzam a esse resultado. De facto, uma solução simbólica traduz o tipo de raciocínio que foi empregue nessa resolução, ou cálculo. No caso de uma expressão como $1+2+3=6$, uma solução possível seria, por exemplo, a sequência

$$(1+2)+3 \downarrow^{1} \rightarrow_{\kappa} 3+3 \rightarrow_{\kappa} 6$$

A resolução simbólica de uma expressão matemática está intimamente ligada à noção que temos de raciocínio. Raciocinar é procurar soluções simbólicas que conduzam aos resultados que pretendemos. Mesmo em casos tão simples como os que acabámos de ver, existem formas diferentes de raciocinar. Por exemplo, uma outra forma de raciocinar sobre essa expressão seria obter esse mesmo resultado através da seguinte solução simbólica,

$$1+(2+3) \downarrow^{1} \rightarrow_{\kappa} 1+5 \rightarrow_{\kappa} 6$$

## 4.5 Conclusões

Neste capítulo, apresentámos o trabalho que desenvolvemos na área da simplificação de expressões matemáticas e que designámos por resolução simbólica de expressões matemáticas, por reescrita axiomática. O trabalho que desenvolvemos

consistiu, essencialmente, no desenvolvimento de uma linguagem computacional de reescrita de expressões matemáticas, baseada na representação lógica dessas expressões, como estruturas triangulares, e na transformação lógica dessas estruturas, por reescrita axiomática, utilizando apenas uma lógica de 1ª ordem.

Uma parte importante desse trabalho foi o desenvolvimento de uma estrutura arbórea binária para a representação lógica de expressões matemáticas e que designámos por estruturas triangulares. Uma estrutura triangular é uma composição lógica de cinco (5) estruturas triangulares básicas. Cada componente possui uma estrutura triangular básica e identifica uma expressão com essa estrutura. Uma estrutura triangular básica define o posicionamento dos argumentos e operadores matemáticos, numa estrutura binária com uma específica assinatura. A representação lógica de expressões matemáticas, como estruturas triangulares, é uma forma de representação que pode ser aplicada a qualquer tipo expressão que seja representável de uma forma arbórea e binária.

A outra parte importante do nosso trabalho foi o aspecto relacionado com a forma como essas expressões podem ser manipuladas computacionalmente com vista à sua simplificação. Nem todas as expressões com esse tipo de estrutura podem ser transformadas por reescrita axiomática. A transformação lógica, por reescrita axiomática, envolve apenas a reescrita de expressões, como estruturas triangulares básicas. Expressões que necessitem de mais do que uma estrutura é no contexto de uma resolução simbólica.

Como vimos, a transformação lógica de expressões matemáticas envolve a aplicação sistemática de um certo número de estratégias de resolução baseadas na leitura operacional dessas expressões, e na aplicação de um certo número de regras de reescrita como base axiomática. Vimos também que aplicação dessas regras e estratégias depende apenas da estrutura triangular dessas expressões.

O Prolog foi a linguagem de implementação que escolhemos para

representar a estrutura triangular e a reescrita axiomática de expressões matemáticas.

# Capítulo 5 A Aplicação Prática

## O Assistente de Matemática

## 5.1 Introdução

O Assistente de Matemática (*MathSolver*) é um programa que desenvolvemos em Prolog, capaz de simplificar, de uma forma simbólica e detalhada, expressões matemáticas definidas num domínio restrito da Álgebra e do Cálculo e que envolvem apenas uma única variável matemática. Essas expressões envolvem apenas operadores matemáticos do tipo que foi definido na Secção 2.2.3. A variável matemática está representada nessas expressões pelo termo atómico $x$. As expressões matemáticas são introduzidas no Assistente de uma forma natural ou quasi-natural.

**Exemplo 5.1-1** ***Escrita de*** $\log_e x + \log_e 2x$

> *A expressão* $\log_e x + \log_e 2x$ *pode ser introduzida, de uma forma quasi-natural, como "sum log e x log e prod 2 x", ou de uma forma natural, em linguagem corrente, como "a soma do logaritmo natural de x com o logaritmo natural do produto de 2 por x", ou de uma forma que seja uma combinação dessas duas.*

---

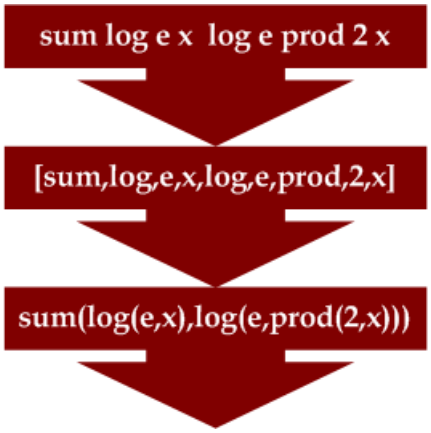
As expressões naturais ou quasi-naturais estão representadas, em Prolog, por listas.

**Exemplo 5.1-2** ***Representação de "sum log e x log e prod 2 x"***

*A expressão "sum log e x log e prod 2 x" é representada, em Prolog, por uma lista do tipo* $[sum, log, e, x, log, e, prod, 2, x]$.

---

A conversão de *strings* para *listas* é efectuada pelo Assistente através da sua interface natural, escrita em Java[53]. Trata-se simplesmente de uma formatação de *strings*. A *lista* é, por sua vez, analisada sintacticamente pelo Assistente e convertida numa expressão lógica, $\text{sum}(\log(e, x), \log(e, \text{prod}(2, x)))$.




O Assistente de Matemática é uma aplicação prática do método de resolução simbólica que defendemos nesta tese, e destina-se essencialmente a pessoas numa fase inicial de aprendizagem da Matemática. Na nossa opinião, a forma natural ou quasi-natural de lidar com expressões matemáticas põe em evidência o significado operacional dessas expressões (Secção 2.3.1). Essa forma de expressão enfatiza também a importância da linguagem natural, na aprendizagem e compreensão da Matemática.

O Assistente lida essencialmente com a simplificação de expressões

[53] Para esta tese, foi utilizada o JBuilder 7 da Borland.

matemáticas, por reescrita axiomática. O próprio cálculo de derivadas é, tratado nesta tese como uma forma de simplificação envolvendo o operador derivada.

O Assistente de Matemática efectua cálculos numéricos, de uma forma simbólica e detalhada. O desempenho do Assistente depende, apenas, dos recursos computacionais que estiverem ao seu dispor (memória e potência do computador) e do grau de complexidade das expressões que se pretende simplificar.

**Exemplo 5.1-3 Cálculo de** $\frac{d^3}{dx^3}\sqrt{x+\sqrt{x}}$

*O cálculo de uma derivada, como* $\frac{d^3}{dx^3}\sqrt{x+\sqrt{x}}$ *, depende do poder redutor da expressão que se pretende derivar. A expressão, neste caso,* $\sqrt{x+\sqrt{x}}$ *, é irredutível. Como resultado, a aplicação de uma regra de derivação vai tornando a expressão cada vez maior o que afecta, por conseguinte, o desempenho da sua resolução.*

---

**Exemplo 5.1-4 Cálculo de** $\frac{d^5}{dx^5}\sqrt{2\sqrt{x}+3\sqrt{x}}$

*No caso de uma expressão, como* $\frac{d^5}{dx^5}\sqrt{2\sqrt{x}+3\sqrt{x}}$ *, o desempenho não é afectado. A expressão, neste caso,* $\sqrt{2\sqrt{x}+3\sqrt{x}}$ *é redutora e, portanto, rapidamente resolvida por reescrita axiomática.*

---

## 5.2 A Linguagem de Expressões

### *5.2.1 A Frase Matemática*

Como em qualquer linguagem de expressões é necessário desenvolver formas eficientes de se escrever e representar frases, nessa linguagem. Como neste caso se trata de uma linguagem que envolve a resolução simbólica de expressões matemáticas, é necessário desenvolver meios de introduzir essas expressões e de exprimir *o que fazer* com elas. Expressões como "*resolver o logaritmo natural do quadrado de x*" exprimem acções onde o objecto é uma expressão matemática. Expressões que nesta tese são designadas por frases matemáticas. Neste contexto, uma expressão matemática como "*logaritmo natural do quadrado de x*", teria o mesmo significado que a frase "*escrever o logaritmo natural do quadrado de x*". Quando se *escreve* uma expressão como $\log_e x^2$, está-se, de facto, a executar uma frase matemática com esse significado operacional. Podíamos dizer que o verbo *escrever* está implícito no significado operacional de qualquer expressão matemática. Para resolver ou executar essas expressões é necessário utilizar outros verbos e, portanto, formar outras frases.

**Definição 5.2-1** ***Frase Matemática***

> *Uma frase matemática é uma expressão, escrita numa linguagem natural, onde está representada, de forma explícita ou implícita, o tipo de acção que se pretende associar ao significado operacional de uma expressão matemática.*

Verbos, como *resolver*, *achar* ou *derivar*, exprimem o tipo de acções que estão, normalmente, associados com essas expressões. Acções que dependem, única e exclusivamente, do significado operacional dessas expressões, ou seja, da sua estrutura triangular. Como é óbvio, as expressões matemáticas podem estar representadas, numa frase, de uma forma explícita ou implícita.

**Exemplo 5.2-1** ***Frase matemática explícita***

*A expressão "resolver o logaritmo natural do quadrado de 4" é uma frase onde a expressão matemática está representada de forma explícita.*

---

Uma frase matemática é representada, em Prolog, por uma lista, ou seja, por um termo do tipo $[Word_i]$, onde $Word_i$ representa a $i$-gésima palavra nessa frase.

**Exemplo 5.2-2** ***Representação de uma frase***

*A expressão "resolver o logaritmo natural do quadrado de 4" é representada, em Prolog por um termo composto do tipo*

$$[resolver, a, expressão, logarítmo, natural, do, quadrado, de, 4]$$

---

Certos operadores matemáticos possuem formas verbais com significados específicos. Expressões como "*derivar a raiz quadrada de x*" ou "*elevar x ao quadrado*" exprimem acções que estão, directamente, relacionadas com o significado operacional dessas expressões. O verbo *derivar*, por exemplo, tem o mesmo significado que a frase "*resolver a derivada de*". É um tipo de acção que poderíamos designar como a forma verbal do operador derivada.

Tal como as expressões matemáticas, também as frases são representadas por expressões lógicas.

**Exemplo 5.2-3** ***Estrutura lógica de uma frase***

*Uma frase como "resolver o logaritmo natural do quadrado de x" possui uma estrutura do tipo*

resolve(practice(expression(log($e$,(pwr(2,$x$))))))

*A assinatura é, neste caso, uma expressão do tipo resolve_practice_expression.*

---

Mas nessa expressão, como se pode ver, estão presentes duas assinaturas, *resolve_practice_expression* e *log_pwr*. A primeira é a assinatura dessa frase e a outra, *log_pwr*, a assinatura da expressão matemática com a qual essa frase está associada.

Tal como os operadores matemáticos, também os verbos possuem estruturas lógicas bem definidas. A assinatura de uma frase define também a sua classe, ou seja, o tipo de estrutura lógica do seu verbo. Uma frase com uma assinatura do tipo *resolve_practice_expression* possui uma estrutura lógica do tipo *resolve*. Verbos como *simplificar*, *achar* e outros, são formas equivalentes do verbo *resolver*.

Com esse tipo de estrutura é possível construir frases como "*resolver o passo seguinte*" ou "*resolver a expressão*". A primeira teria uma estrutura lógica do tipo resolve(practice(step(*next*))) e a segunda, uma estrutura do tipo resolve(practice(*expression*)). Em ambas, como se pode observar, a expressão matemática que se pretende resolver, está representada de forma implícita.

### *5.2.2 O Processamento de Frases*

A resolução simbólica de expressões matemáticas é feita utilizando uma linguagem, natural ou quasi-natural, cujas frases matemáticas são escritas, em Português. Essas frases são introduzidas no Assistente através de uma interface gráfica (Secção 5.3). As frases matemáticas são analisadas e processadas através da resolução de literais baseados no predicado query(*Phrase*), onde *Phrase* é uma

frase matemática.

O processamento de uma frase matemática (*query*) é simplesmente o resultado da sua análise e a consequente execução da sua representação lógica.

A implementação Prolog do predicado ***query*** / 1 está listada na Secção A.2.1 do Apêndice A.

A análise de uma frase matemática (*parse*) pode ser descrita pelo seguinte algoritmo

***se*** *a frase é válida* ***então***
    *executar a frase*
***senão***
    *rejeitar a frase*

A análise é baseada no predicado ***parse*** / 5 . A implementação Prolog desse predicado está listada na Secção A.2.1 do Apêndice A.

**Exemplo 5.2-4** ***Execução de uma frase***

*Uma frase do tipo* resolve *é representada por uma expressão lógica do tipo* resolve(practice(expression(*Arg*))) . *A formação dessa frase é simplesmente o resultado de uma análise sintáctica dos seus constituintes e da sua composição lógica, através da aplicação de regras gramaticais baseadas no predicado* ***s*** / 1 .

---

A execução dessa frase (*process*) pode ser descrita através do seguinte algoritmo

***se*** *a frase é do tipo* resolve ***então***
    ***se*** *a expressão está definida* ***então***

*resolver o passo*
*visualizar a solução*
***senão***
*definir a expressão*
*resolver o 1º passo*
*visualizar a prática*
***senão***
*rejeitar a frase*

A execução de uma frase matemática é baseada no predicado ***process*** / 5 . A implementação Prolog desse predicado está listada na Secção A.2.1 do Apêndice A.

## 5.3 A Interface Gráfica do Assistente

O Assistente de Matemática que desenvolvemos utiliza uma escrita em linguagem natural ou quasi-natural que evidencia o significado operacional da expressão que se pretende resolver simbolicamente. A forma como uma expressão é escrita põe em evidência não só a sua assinatura como também a estrutura triangular que está subjacente a essa assinatura (Secção 2.4.1).

**Exemplo 5.3-1** ***Escrita de*** $1+2+3$

> *Uma expressão como* $1+2+3$ *deve ser lida, operacionalmente, como* $(1+2)+3$ *ou* $1+(2+3)$. *Tanto uma como a outra possuem uma assinatura* sum_sum . *As suas estruturas são, porém, de níveis diferentes. A primeira é escrita como "soma da soma de 1 2 e 3" e a segunda como "soma de 1 com a soma de 2 com 3".*

---

Ambas são formas correctas de se ler essa expressão. Cada uma dessas leituras conduz a um raciocínio ou estratégia de resolução diferente. Os parênteses

são apenas uma forma simbólica de evidenciar a estrutura que está subjacente a essas expressões.

**Exemplo 5.3-2** ***Ordem de leitura dos operadores***

> *Numa expressão, como* $(1+2)+3$, *o operador soma com âmbito mais alargado é o operador que está mais à direita e, portanto, aquele que deve ser lido primeiro.*

---

Escrita em linguagem corrente, a expressão possui uma leitura inequívoca. Essa leitura exprime apenas uma forma diferente de se raciocinar sobre essa expressão. As diferentes formas de leitura que uma expressão pode exibir não são mais do que formas equivalentes de se representar essa mesma expressão. Essas leituras estão, de certo modo, relacionadas com o facto de alguns operadores serem, por natureza, associativos.

**Exemplo 5.3-3** ***Escrita de*** $x+(x+2)$

> *Uma expressão, como* $x+(x+2)$, *escrita como "soma de x com a soma x com 2", teria de ser transformada, associando primeiro os termos semelhantes* $x$, $(x+x)+2$, *e só depois reduzida, por via axiomática (Secção 4.3.1.2). O resultado seria a expressão equivalente,* $2x+2$. *Escrita, porém, como "soma da soma de x com x com 2", teria o mesmo efeito mas sem o passo associativo.*

---

As várias leituras não são mais do que formas diferentes de se compor essas expressões.

**Exemplo 5.3-4** ***Escrita de*** $\log_e \sqrt{4^3}$

*Expressões, como* $\log_e \sqrt{4^3}$ *, possuem uma única leitura operacional. A expressão só pode ser escrita como "logaritmo natural da raiz quadrada do cubo de 4" ou de uma forma mais abreviada como, por exemplo, "log e root 2 pwr 3 4".*

---

Neste caso, existe apenas uma forma de compor essa expressão.

Mas, seja qual for o número de leituras que se podem fazer de uma expressão, a sua resolução simbólica depende também da forma como as suas componentes são lidas ou interpretadas. Neste caso, as duas componentes seriam interpretadas como o "*logaritmo de uma raiz*" e a "*raiz de uma potência*".

Escrever expressões, em linguagem corrente, pode causar algum desconforto. Porém, essa é a única forma correcta de comunicá-las oralmente. A alternativa no caso de uma expressão, como $\log_e \sqrt{4^3}$ , seria utilizar formas como $\ln(\text{sqrt}(4 \wedge 3))$ . Mas, como vimos na Secção 2.3.2, essa forma de escrita pode introduzir ambiguidades e necessitar do uso de parênteses para a desambiguar.

**Exemplo 5.3-5** ***Escrita quasi-natural de*** $\log_e \sqrt{4^3}$

*A expressão* $\log_e \sqrt{4^3}$ *pode ser escrita, de uma forma mais compacta, como "log e raiz 2 potência 3 4".*

---

Como veremos mais à frente existem formas de reduzir esse desconforto

ainda mais através do uso de ferramentas gráficas.

A forma de escrita natural ou quasi-natural proporciona, também, uma forma correcta de exprimir a "*ordem de precedência*" dos operadores. Numa expressão matemática, os operadores estão contextualizados pela estrutura em que estão inseridos.

**Exemplo 5.3-6** ***Ordem de precedência***

*Numa expressão, como* $(1+2)+3$*, a soma* $1+2$ *está contextualizada pela soma* $(\bullet)+3$*. A soma* $1+2$ *tem precedência sobre esta.*

---

Aprender como escrever $\log_e \sqrt{4^3}$ deve passar primeiro pela escrita do seu significado operacional (Secção 2.3.1).

A forma de escrita gráfica, e mais compacta, da Matemática, que muitos classificam de *natural*, não é assim tão fácil de aprender como muitos poderão pensar à priori. Os alunos, especialmente aqueles com maiores dificuldades em Matemática, sentem uma grande dificuldade em lidar com ela. Por um lado, porque ainda a não compreendem muito bem; por outro, porque ainda não estão suficientemente habituados à sua forma "*não convencional*" de escrita.

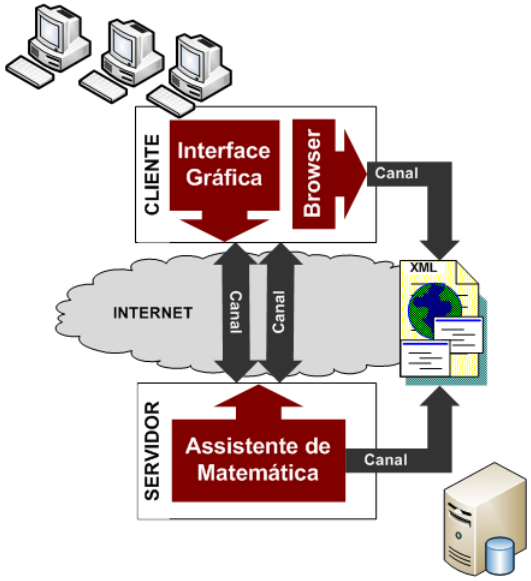



**Figura 5.3.1** *Interface natural do Assistente de Matemática*

Um dos objectivos do Assistente é, precisamente, tornar a linguagem da Matemática mais acessível aos utilizadores, do ponto de vista operacional. O Assistente reconhece a escrita de expressões, em linguagem corrente, e apresenta os resultados de uma resolução simbólica, em notação matemática, estabelecendo assim uma ponte entre a *linguagem corrente* e a *notação matemática*, ou seja, entre a *leitura* e *escrita* de expressões matemáticas. Assim, quando o utilizador escreve uma expressão, em linguagem corrente, como por exemplo, "*logaritmo natural da raiz quadrada do cubo de 4*", o que aparece escrito é a expressão $\log_e \sqrt{4^3}$ . Para apresentar as expressões, em notação matemática, o Assistente converte as expressões lógicas em MathML e disponibiliza-as através do *browser*.

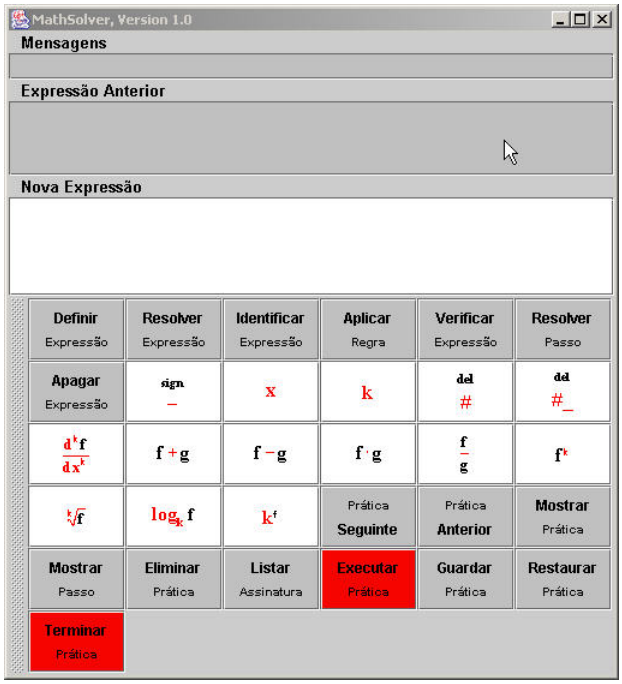



**Figura 5.3.2** ***Assistente de Matemática***

Para facilitar a escrita de expressões matemáticas, em linguagem corrente, ou numa versão mais abreviada dessa linguagem (quasi-natural), o Assistente disponibiliza um conjunto de ferramentas gráficas que tornam essa escrita mais fácil. Essas ferramentas permitem que a escrita de expressões matemáticas seja feita de acordo com a sua leitura operacional.

A interface gráfica (*MathClient*) do Assistente está ilustrada na Figura 5.3.1. Essa interface foi desenvolvida em linguagem Java e comunica com o Assistente através de "*sockets*". O Assistente é instanciado por um Servidor de Assistentes (*MathServer*). O servidor é também um programa escrito em Java. O protótipo que desenvolvemos escuta apenas num determinado porto TCP[54]. Na versão, que é apresentada nesta tese, o servidor apenas instancia apenas um Assistente de cada vez. O servidor de Assistentes poderá instanciar, no futuro, mais

[54] Protocolo de transmissão de dados com confirmação de recepção de dados (***T**ransmission **C**ontrol **P**rotocol*)

do que um Assistente e partilhá-lo com vários clientes. Mas para isso, é necessário fazer algumas alterações ao Assistente e ao servidor de Assistentes, o que está fora do âmbito desta tese. Para o desenvolvimento deste protótipo, optámos pelo modelo cliente-servidor por ser esse o modelo que melhor se adequa a este tipo de sistemas. As interfaces gráficas podem ser disponibilizadas localmente, e os Assistentes, remotamente, através da Internet ou de uma intranet. O protótipo (*MathClient*, *MathSolver* e *MathServer*) , que desenvolvemos para esta tese, foram instalados na mesma máquina, mas comunicam entre si através de um servidor Tomcat/Apache.

Com este tipo de interface é possível escrever expressões matemáticas conhecendo apenas a ordem de precedência dos vários operadores presentes nessas expressões, isto é, sabendo como ler expressões.

**Exemplo 5.3-7** ***Como utilizar a interface gráfica***

*Para escrever uma expressão logaritmo, basta premir no botão que representa esse operador* $\log_k f$ *, e escrever, de seguida, os seus dois argumentos. O Assistente escreve a expressão "log # _", como um "template", que evidencia não só o tipo de expressão que se pretende escrever, como assinala também os locais onde os seus argumentos devem ser inseridos.*

---

Com este tipo de ferramentas gráficas, consegue-se não só reduzir a probabilidade de se cometerem erros com este tipo de escrita, como também tornar a escrita, em linguagem corrente, mais apelativa.

Trata-se de uma forma de escrita que poderíamos classificar de "*semi-inteligente*". Com esse tipo de escrita é possível reconhecer certas posições como

posições chave[55], e permitir assim o seu preenchimento de uma forma que consideramos mais inteligente. Essas posições podem ser preenchidas, em qualquer ordem, utilizando o teclado ou as outras ferramentas gráficas, disponíveis nessa interface. O símbolo "#", por exemplo, é utilizado como marcador de posições-chave para argumentos numéricos. O símbolo "_", por outro lado, é utilizado como marcador de posições-chave para qualquer outro tipo de argumento, incluindo numéricos.

Por exemplo, para escrever a expressão $\log_e \sqrt{4^3}$, em linguagem corrente, mas de forma abreviada (quasi-natural), poderíamos fazê-lo premindo um certo número de ferramentas gráficas na seguinte sequência

| ***Ferramenta*** | ***Frase*** | ***Ref[a]*** |
|---|---|---|
| [*definir*] | definir a expressão _ | (1) |
| [*log*] | definir a expressão log # _ | (2) |
| [*root*] | definir a expressão log # root # _ | (3) |
| [*pwr*] | definir a expressão log # root # pwr # _ | (4) |
| [*e*] | definir a expressão log e root # pwr # _ | (5) |
| [2] | definir a expressão log e root 2 pwr # _ | (6) |
| [3] | definir a expressão log e root 2 pwr 3 _ | (7) |
| [4] | definir a expressão log e root 2 pwr 3 4 | (8) |

Premindo o botão "*definir expressão*", consegue-se escrever o início dessa frase matemática (Secção 5.2.1) (1). O símbolo " _ " indica a posição onde o resto

[55] Estas posições são, normalmente, designadas, em inglês, por "*place-holders*".

da frase deve ser inserido. Como se trata de uma expressão logaritmo, o primeiro operador a ser escrito é o operador logaritmo. Para o escrever, basta premir na ferramenta $\log_k f$. O Assistente insere nessa posição o "*template*" $\log \# \_$. Fazendo o mesmo para os restantes operadores, obtemos a escrita de um "*template*" para essa expressão $\log \# \text{root} \# \text{pwr} \# \_$, com algumas posições-chave ainda por preencher (4). Para posicionar o cursor na primeira posição chave # (5), o botão $k$ deve ser premido. A base do logaritmo é então digitada utilizando o teclado. Premindo novamente no botão $k$, faz com que o cursor seja reposicionado na próxima posição chave # (6). Neste caso, é o grau da raiz que é digitado. Procedendo desta forma, consegue-se escrever uma expressão, em linguagem quasi-natural, digitando apenas alguns números e/ou letras.

Uma vez definida a expressão, é necessário analisá-la. Para isso é necessário envolver o Assistente. Para se comunicar com o Assistente, é necessário enviar-lhe frases com objectivos bem definidos. Essas frases são as frases matemáticas que discutimos na Secção 5.2.1. Tal como as expressões, as frases matemáticas também podem ser construídas utilizando as ferramentas gráficas disponibilizadas na interface.

**Exemplo 5.3-8** ***Definir a expressão***

*Para definir uma expressão é necessário enviar o Assistente uma frase do tipo "Definir Expressão".*

---

As frases são enviadas ao Assistente premindo o botão "*Executar Prática*". Uma prática é simplesmente uma sessão prática envolvendo a resolução simbólica, total ou parcial, de uma expressão. Para obter soluções simbólicas diferentes, para uma mesma expressão, é necessário definir e resolver essa expressão em práticas diferentes. O Assistente reconhece um certo número de frases. Todas elas são

iniciadas por um verbo.

O Assistente reconhece os seguintes verbos:

- **Definir** – Este verbo é interpretado pelo Assistente como um pedido para iniciar uma sessão prática para uma expressão. Por exemplo, a frase "*definir a expressão logaritmo natural do quadrado de 4*", é uma frase deste tipo;
- **Resolver** - Este verbo é interpretado pelo Assistente como um pedido para resolver, total ou parcialmente, uma determinada sessão prática. Por exemplo, a frase "*resolver o passo seguinte*", é uma frase deste tipo;
- **Identificar** – Este verbo é interpretado pelo Assistente como um pedido para identificar uma determinada componente (subexpressão) da expressão. Por exemplo, a frase "*identificar o operador derivada 2*" (isto é, identificar a segunda das derivada na expressão), é uma frase deste tipo;
- **Aplicar** – Este verbo é interpretado pelo Assistente como um pedido para aplicar uma regra (axioma ou teorema) a uma determinada componente da expressão. Por exemplo, a frase "*aplicar a regra relate_log_pwr ao logaritmo 2*" (isto é, aplicar a relação fundamental com a assinatura *log_pwr* ao segundo dos logaritmos na expressão), é uma frase deste tipo;
- **Verificar** – Este verbo é interpretado pelo Assistente como um pedido para verificar se uma expressão é ou não equivalente. Por exemplo, a frase "*verificar a expressão produto de 8 por x*", é uma frase deste tipo;
- **Listar** – Este verbo é interpretado pelo Assistente como um pedido para listar as regras (axiomas ou teoremas) com uma determinada assinatura. Por exemplo, a frase "*listar a assinatura log*", é uma frase deste tipo;

- **Mostrar** - Este verbo é interpretado pelo Assistente como um pedido para mudar para uma determinada sessão prática. Por exemplo, a frase "*mostrar a prática 2*", é uma frase deste tipo.

Uma sessão prática é iniciada e conduzida pelo utilizador através da interface do Assistente. O diálogo com o Assistente é mantido enviando frases deste tipo e observando o resultado da interacção no browser que está instalado no computador. O utilizador comunica com o Assistente através da interface. O Assistente, por outro lado, comunica com o utilizador através da interface e do *browser*. Na versão desenvolvida para esta tese, as únicas mensagens enviadas pelo Assistente e disponibilizadas na interface, são mensagens de estado. No futuro, pensamos poder incluir outras mensagens, como por exemplo, a activação e desactivação de ferramentas ou a interacção directa do Assistente na identificação de componentes ou detecção de gestos.

A seguir são descritas as frases que podem ser enviadas ao Assistente. As frases são analisadas e processadas pelo Assistente (Secção 5.2.2) e o resultado enviado para o *browser* sob a forma de um ficheiro XML.

### *5.3.1 Definir uma Expressão*

A resolução simbólica de expressões matemáticas, envolve vários passos. O primeiro é obviamente definir a expressão que se pretende resolver com o auxílio do Assistente. Para definir uma expressão, é necessário iniciar uma sessão prática. Uma sessão prática é iniciada enviando ao Assistente uma frase matemática do tipo

definir [a] [expressão] ***Expressão***

A presença de uma ***Expressão*** nessa frase faz com que o Assistente associe essa expressão a essa sessão prática.

**Exemplo 5.3-9** ***Definir a expressão*** $\frac{d^2}{dx^2}\log_e x^2$

*Uma frase como "definir a expressão der 2 log e pwr 2 x" inicia uma sessão prática para a* ***Expressão***, $\frac{d^2}{dx^2}\log_e x^2$. *Essa expressão pode ser escrita nessa forma abreviada (quasi-natural), ou em linguagem corrente, como "definir a segunda derivada do logaritmo natural do quadrado de x". A escrita, em linguagem corrente, só pode ser feita através do teclado. O reenvio dessa frase faz com que a expressão seja redefinida.*

---

Se omitirmos a ***Expressão***, o Assistente assume que se trata de uma redefinição da expressão que está activa nessa altura. A frase "*definir a expressão*" inicia uma nova sessão prática para a expressão que está a ser resolvida, nessa altura. Redefinir uma expressão faz com que essa expressão possa ser resolvida de uma maneira diferente. O reenvio dessa frase faz com que a expressão activa seja redefinida.

Cada uma das sessões iniciadas é resolvida independentemente mas com o total conhecimento do que foi feito nas outras práticas iniciadas para essa mesma expressão. Assim, se numa determinada sessão prática for aplicada uma determinada regra, nas outras, essa regra não pode ser aplicada, automaticamente, no mesmo contexto. O Assistente terá de aplicar outras regras caso elas existam.

**Exemplo 5.3-10** ***Aplicação da regra*** *relate_log_pwr*

*Se numa sessão prática iniciada para uma expressão, como* $\log_e 4^2$, *for aplicada uma regra de reescrita por transformação, do tipo* relate_log_pwr

[56], $\log_e 4^2 \rightarrow 2\log_e 4$, *então numa outra sessão, iniciada para essa mesma expressão, a mesma regra não pode ser aplicada. O Assistente terá de aplicar uma outra, como por exemplo, uma regra de redução por cálculo do tipo* *evaluate_pwr* [57], $\log_e 4^2 \rightarrow \log_e 16$, *ou uma regra de reescrita por transformação do tipo* *factor_out_pwr* [58], $\log_e 4^2 \rightarrow \log_e (2^2)^2$.

### *5.3.2 Resolver uma Prática por Completo*

Uma prática pode ser resolvida por completo pelo Assistente. Uma prática pode ser completada a partir de qualquer passo da resolução. Uma prática é resolvida por completo enviando ao Assistente uma frase do tipo

resolver [a] prática

Essa frase faz com que a prática que esteja activa, nessa altura, seja finalizada. O Assistente resolve automaticamente todos os passos que estão ainda por resolver.

Se numa determinada prática, o último passo resolvido foi o $i$-*gésimo* passo, então o envio desta frase faz com que os restantes passos, do $i+1$-*gésimo* em diante, sejam resolvidos, e a prática finalizada. Essa frase faz com que o resultado, obtido no último passo, seja resolvido, por completo. O reenvio dessa frase não produz qualquer efeito, pois a expressão já está resolvida. Para se obter uma outra solução para essa mesma expressão, é necessário redefini-la (ver "Definir expressão").

[56] Isto é, aplicar a relação fundamental entre logaritmos e potências.

[57] Isto é, resolver, numericamente, a potência.

[58] Isto é, factorizar a base da potência.

### *5.3.3 Resolver uma Prática Passo a Passo*

Uma prática pode ser resolvida passo a passo pelo Assistente. Quando resolvida pelo Assistente, a resolução é feita de forma automática utilizando a estratégia sugerida pelo Assistente. Como veremos mais à frente, uma prática pode ser resolvida passo a passo pelo utilizador. Uma prática é resolvida, passo a passo, pelo Assistente, enviando ao Assistente uma frase do tipo

resolver [o] passo seguinte

Essa frase faz com que o Assistente aplique uma regra de reescrita (axioma ou teorema) à expressão obtida no passo anterior. A regra é seleccionada pelo Assistente com base na assinatura dessa expressão. O alvo é a prática que está activa nessa altura. O novo passo é adicionado à prática e o resultado apresentado no *browser* como o próximo passo a resolver. O reenvio dessa frase faz com que cada passo seja resolvido a partir do anterior. Para se resolver uma prática, passo a passo, de forma automática, basta enviar uma frase desse tipo, ao Assistente, para cada passo que esteja ainda por resolver. O reenvio dessa frase deixa de ter efeito depois de a prática estar resolvida. O utilizador não intervém na resolução.

### *5.3.4 Identificar uma Componente*

Para intervir numa resolução, o utilizador tem de ser capaz de identificar a componente sobre a qual pretende aplicar a regra. Uma expressão pode conter uma ou mais componentes (subexpressões). Cada componente é identificada pelo seu operador principal e pela ordem desse operador na expressão. O operador principal (Secção 2.2.1) é o operador que está listado mais à esquerda na assinatura dessa componente. Por exemplo, o operador *log* é o operador principal numa assinatura *log_pwr*. Uma componente pode ser identificada, enviando ao Assistente uma frase do tipo

identificar [o] [operador] ***Operador Ordem***

**Exemplo 5.3-11** ***Identificar um operador em*** $\frac{\mathrm{d}}{\mathrm{dx}}(2x^2)+\frac{\mathrm{d}}{\mathrm{dx}}(3x)$

*Para identificar a segunda derivada numa expressão, como* $\frac{\mathrm{d}}{\mathrm{dx}}(2x^2)+\frac{\mathrm{d}}{\mathrm{dx}}(3x)$*, com várias derivadas, utiliza-se a frase "identificar o operador der 2". Esta frase identifica a componente cujo* ***Operador*** *principal (*$\mathsf{der}$*) é a segunda (****ordem****) derivada (*$\mathsf{der}$*) nessa expressão. Na expressão,* $\frac{\mathrm{d}}{\mathrm{dx}}(2x^2)+\frac{\mathrm{d}}{\mathrm{dx}}(3x)$*, estão presentes duas derivadas (*$\mathsf{der}$*). A componente* $\frac{\mathrm{d}}{\mathrm{dx}}(3x)$ *seria a componente identificada como a segunda derivada (*$\mathsf{der\ 2}$*) nessa expressão. A expressão alvo é a expressão activa nessa altura. A expressão activa é a expressão que foi obtida como resultado do último passo dessa prática.*

---

Por convenção, a definição de uma expressão é o passo inicial (ou passo 0) de uma resolução.

O Assistente não identifica apenas a componente. Identifica também a sua assinatura.

**Exemplo 5.3-12** ***Assinatura de*** $\frac{\mathrm{d}}{\mathrm{dx}}(2x^2)+\frac{\mathrm{d}}{\mathrm{dx}}(3x)$

*A componente* $\frac{\mathrm{d}}{\mathrm{dx}}(3x)$ *é identificada como a segunda derivada na expressão* $\frac{\mathrm{d}}{\mathrm{dx}}(2x^2)+\frac{\mathrm{d}}{\mathrm{dx}}(3x)$*, e aquela cuja assinatura é* $\mathsf{der\_prod}$*. A outra,* $\frac{\mathrm{d}}{\mathrm{dx}}(2x^2)$*, seria identificada como a primeira nessa expressão e aquela*

*cuja assinatura é também* $der\_prod$ .

---

O reenvio dessa frase não produz qualquer outro efeito. A componente é simplesmente identificada de novo.

A ordem de um operador numa expressão é precisamente a ordem em que esse operador é lido numa leitura operacional dessa expressão.

**Exemplo 5.3-13** ***Ordem de um operador em*** $(x+2x)+(3x+x)$

*Numa expressão, como* $(x+2x)+(3x+x)$, *a segunda soma é a expressão* $x+2x$. *Essa expressão é, de facto, a segunda soma na expressão "a soma da soma de x com o produto de 2 por x com a soma da soma do produto de 3 por x com x".*

---

## *5.3.5 Aplicar uma Regra de Reescrita*

Para intervir numa resolução, o utilizador tem de saber que regras deve aplicar e onde as deve aplicar. Isso pressupõe, naturalmente, que o utilizador seja capaz de identificar as componentes sobre as quais pretende aplicar essas regras (axiomas ou teoremas). As regras são identificadas pelo seu tipo de reescrita (Secção 4.3.1) e pela sua classe (Secção 2.4.1). Por exemplo, uma regra de reescrita por transformação (Secção 4.3.1.4), pertencente à classe $log\_pwr$, é designada por $relate\_log\_pwr$ [59] e pode ser aplicada a expressões com assinaturas $log\_pwr$.

Como vimos na Secção 2.4.1, a assinatura de uma expressão define a sua

[59] Uma reescrita por conversão axiomática inclui não só a aplicação de relações fundamentais (regras do tipo *relate*) mas também a aplicação de propriedades como a associatividade (regras do tipo *associate*) e outras (ver Secção 4.3.1.4).

classe de equivalência Uma expressão pode ser reescrita, axiomaticamente, enviando ao Assistente uma frase do tipo

aplicar [a] [regra] ***Regra*** [ao] [operador] ***Operador Ordem***

A ***Regra*** especificada nessa frase é aplicada à componente identificada pelo ***Operador*** nessa ***Ordem***. A ordem de um operador é determinada pela leitura operacional dessa expressão.

**Exemplo 5.3-14** ***Aplicar a regra*** *relate_log_pwr*

*Na frase "aplicar a regra relate_log_pwr ao operador log 2", uma regra de reescrita (relate_log_pwr) do tipo relate (aplicação de uma relação fundamental), da classe log_pwr (logaritmo de potências), é aplicada à segunda (2) componente derivada (der) da expressão alvo. Essa componente é identificada pelo seu operador principal (der) e pela ordem desse operador, nessa expressão. A componente, neste caso, possui uma assinatura log_pwr.*

---

O reenvio dessa frase pode fazer com que essa mesma regra seja aplicada a uma outra componente com essas mesmas características. Note que a expressão alvo é outra. O reenvio só deve ser feita depois de a componente ter sido devidamente identificada. Essa identificação pode ser feita, como vimos atrás, com a ajuda do Assistente (através do envio de uma frase do tipo "*identificar*").

### *5.3.6 Verificar uma Expressão*

O utilizador pode tentar resolver um ou mais passos, *off-line*, e pedir ao Assistente que verifique o resultado. Os resultados verificados pelo Assistente não precisam de estar numa ordem específica. Isto é, uma sequência de resultados

verificados pelo Assistente não constitui, necessariamente, uma cadeia de derivação. Uma expressão pode ser verificada pelo Assistente, como sendo equivalente à expressão que está a ser resolvida, enviando ao Assistente uma frase do tipo

verificar [a] [expressão] ***Expressão***

A ***Expressão*** é verificada no contexto da prática que está a ser resolvida. A verificação é feita comparando essa expressão com todas as formas equivalentes da expressão que está a ser resolvida. Essas expressões são obtidas pelo Assistente por reescrita axiomática. Como é óbvio, nem todas as expressões equivalentes podem ser verificadas por este método. Só o podem aquelas que forem obtidas por reescrita axiomática.

**Exemplo 5.3-15** ***Verificar*** $\frac{3}{6}$

*A fracção* $\frac{3}{6}$ *não é verificável como sendo uma expressão equivalente a* $\frac{5}{10}$. *Ambas são, de facto, equivalentes a* $\frac{1}{2}$, *mas não são transformáveis, uma na outra, por reescrita axiomática. Para verificar esse tipo de equivalência, seria necessário resolver ambas. Ou seja verificar que ambas se reduzem a* $\frac{1}{2}$.

---

Na presente versão, o resultado que se pretende verificar não é resolvido por reescrita axiomática.

**Exemplo 5.3-16** ***Verificar*** $2\log_e 4$

*A frase "verificar a expressão prod 2 log e 4" verifica apenas se a expressão*

*$2\log_e 4$ é uma das formas equivalentes da expressão $\log_e 4^2$, obtidas por reescrita axiomática. O reenvio dessa frase produz sempre o mesmo efeito, ou seja, se essa expressão é, ou não, uma reescrita axiomática de $\log_e 4^2$.*

---

A verificação de resultados não impõe nenhuma ordem específica de verificação.

**Exemplo 5.3-17** ***Verificar*** $2\log_e 2^2$

*A expressão $2\log_e 2^2$ pode ser verificada depois da expressão $4\log_e 2$ ter sido obtida. Ambas são formas equivalentes da expressão $\log_e 4^2$, obtidas por reescrita axiomática. Porém, não é possível obter a expressão $2\log_e 2^2$ por uma reescrita axiomática de $4\log_e 2$.*

---

### *5.3.7 Listar uma Assinatura*

Para se aplicar uma regra de reescrita, é necessário identificá-la. Como sabemos, as regras de reescrita são identificadas pelo seu tipo de reescrita (Secção 4.3.1) e pela sua classe (Secção 2.4.1). As regras de reescrita estão associadas às assinaturas e podem ser listadas por assinatura. Pode-se obter essa lista enviando ao Assistente uma frase do tipo

listar [a] [assinatura] ***Assinatura***

**Exemplo 5.3-18** ***Listar as regras de*** *log*

*A frase "listar a assinatura log" lista todas as regras associadas à assinatura log .*

---

O reenvio dessa frase produz sempre o mesmo efeito.

### *5.3.8 Mostrar uma Prática*

Apenas uma prática pode estar activa de cada vez. Para mudar de prática e, portanto, tornar uma outra activa, deve-se enviar ao Assistente uma frase do tipo

mostrar [a] [prática] ***Prática***

**Exemplo 5.3-19** ***Mostrar uma prática***

*A frase "mostrar a prática 1" torna a prática 1 activa.*

---

Quando uma prática é activada, a resolução é automaticamente retomada a partir do último passo que foi resolvido. O reenvio dessa frase não produz qualquer efeito, pois a prática já está activa.

## 5.4 A Resolução Simbólica de uma Expressão Matemática

A resolução simbólica de uma expressão matemática começa, como é óbvio, com a definição dessa expressão, ou seja, com a afectação dessa resolução a uma prática. Nesta versão do Assistente, as práticas são apenas enumeradas sequencialmente. Suponhamos então que o utilizador, neste caso, é um aluno com algumas dificuldades em Matemática, que pretende resolver a expressão

$\frac{d^2}{dx^2}x^3 + \frac{d}{dx}x^2$, com a ajuda do Assistente.

O primeiro passo a dar é, portanto, definir essa expressão. Como vimos atrás, uma expressão é definida enviando ao Assistente uma frase do tipo "*definir*". O utilizador prima então o botão "*Definir Expressão*", e a frase "*definir a expressão* _" aparece escrita na janela de *input* do Assistente. Nessa frase não está ainda escrita nenhuma expressão matemática. O utilizador necessita de a completar. Como vimos atrás, uma expressão matemática pode ser escrita utilizando as ferramentas gráficas do Assistente. Neste caso, a sequência de escrita foi a seguinte:

| ***Ferramenta*** | ***Frase*** | ***Ref[a]*** |
|---|---|---|
| [*definir*] | definir a expressão _ | (1) |
| [*sum*] | definir a expressão sum _ _ | (2) |
| [*der*] | definir a expressão sum der # _ _ | (3) |
| [*pwr*] | definir a expressão sum der # pwr # _ _ | (4) |
| [*x*] | definir a expressão sum der # pwr # x _ | (5) |
| [*der*] | definir a expressão sum der # pwr # x der #_ | (6) |
| [*pwr*] | definir a expressão sum der # pwr # x der # pwr # _ | (7) |
| [*x*] | definir a expressão sum der # pwr # x der # pwr # x | (8) |
| [2] | definir a expressão sum der 2 pwr # x der # pwr # x | (9) |
| [3] | definir a expressão sum der 2 pwr 3 x der # pwr # x | (10) |
| [1] | definir a expressão sum der 2 pwr 3 x der 1 pwr # x | (11) |

| | | |
|---|---|---|
| [2] | definir a expressão sum der 2 pwr 3 x der 1 pwr 2 x | (12) |

Uma vez completada, a frase "*definir a expressão sum der 2 pwr 3 x der 1 pwr 2 x*" é enviada ao Assistente premindo o botão "*Executar Prática*". O Assistente analisa a expressão (Secção 5.2.2) e inicia uma prática para essa expressão que nomeia de *prática_1*. Os resultados dessa prática são mantidos num ficheiro XML com esse nome, *pratica_1.xml* . O conteúdo desse ficheiro é listado, automaticamente, no browser desse utilizador.

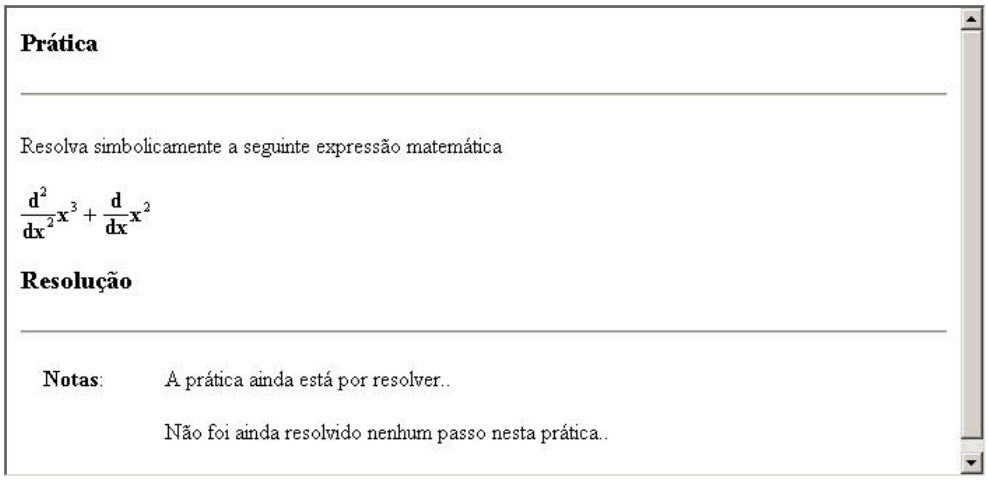



**Figura 5.4.1** ***Início de uma prática***

A Figura 5.4.1 mostra o conteúdo dessa prática depois da expressão $\frac{d^2}{dx^2}x^3+\frac{d}{dx}x^2$ ter sido definida pelo Assistente. A expressão aparece escrita em notação matemática. O Assistente informa também sobre o estado dessa prática. Note que o utilizador podia ter optado por escrever essa expressão em linguagem corrente, ou seja, como "*a soma da segunda derivada do cubo de x com a primeira derivada do quadrado de x*". O resultado seria o mesmo. O utilizador pode testar, desta forma, a sua capacidade de leitura de expressões matemáticas.

A prática está agora iniciada e, portanto, o utilizador pode prosseguir com a

sua resolução. O utilizador pode optar por resolvê-la de uma só vez ou resolvê-la, passo a passo, com ou sem a ajuda do Assistente. Como vimos atrás, para a resolver de uma só vez, bastaria apenas enviar ao Assistente a frase "*resolver a prática*". Esta não é opção que o utilizador decide escolher. Ele decide resolvê-la, passo a passo, sem a ajuda do Assistente. Ele sabe que em qualquer altura pode pedir o auxílio do Assistente. Ele sabe também que, para resolver essa expressão, vai ter de seleccionar componentes e aplicar um certo número de regras (axiomas ou teoremas).

O utilizador decide então aplicar uma regra de derivação à primeira das derivadas. Ele necessita, portanto, de identificar essa componente. Como não se sente muito seguro, decide primeiro certificar-se de que a derivada que escolheu é, de facto, a primeira nessa expressão. Envia ao Assistente a frase "*identificar o operador der 1*". O Assistente identifica o primeiro (*1*) operador derivada (*der*) nessa expressão e envia o resultado dessa identificação para o *browser* do utilizador. O resultado é colocado num ficheiro XML, *expression.xml*, e a componente identificada numa cor diferente. Nesse ficheiro está identificada a assinatura da componente (Secção 2.4.1) e o respectivo contexto, ou seja, o lado esquerdo da fórmula que lhe pode ser aplicada. A Figura 5.4.2 mostra o conteúdo desse ficheiro, tal como ele é disponibilizado no *browser*.

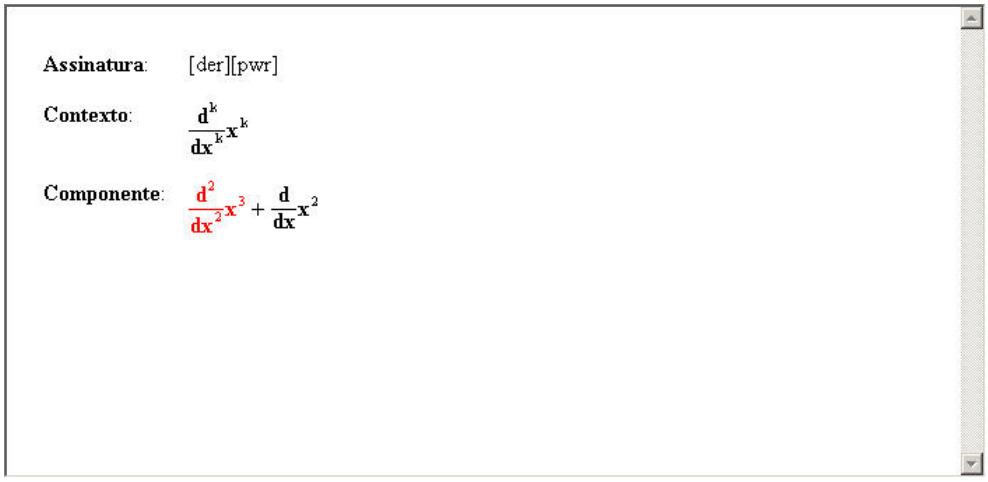



**Figura 5.4.2** ***Identificação de uma componente***

O utilizador pode testar a sua capacidade de leitura, identificando as várias componentes dessa expressão. O significado operacional de uma expressão (Secção 2.3.1) é obtido através desse tipo de leitura. Por exemplo, numa expressão como $(2x)\log_e x$, o primeiro operador produto (*prod 1*) é aquele que é lido como segundo, da esquerda para a direita, ou seja, $(\ldots)\log_e x$. O produto $2x$ é o segundo essa expressão. Os parênteses numa expressão determinam a estrutura dessa expressão. A sua equivalente por associatividade, $2(x\log_e x)$, teria uma estrutura diferente. A primeira seria lida como "*o produto do produto de 2 por x pelo logaritmo natural de x*", ou mais abreviadamente, como "*prod prod 2 x log e x*". A segunda seria lida como "*o produto de 2 pelo produto de x pelo logaritmo natural de x*" ou "*prod 2 prod x log e x*". Em ambas estão bem definidas as ordens de precedência desses operadores.

Vamos supor então que o utilizador consegue identificar a expressão mas que não está ainda suficientemente familiarizado com o conceito de derivação e que não sabe, portanto, que regra deve aplicar nesse caso. Como sabe que se trata de uma derivada, ele pede ao Assistente que liste todas as regras com essa assinatura, ou seja, envia-lhe a frase "*listar a assinatura der*". O Assistente coloca essa

informação no ficheiro *signature.xml* e disponibiliza-a no *browser*. A Figura 5.4.3 mostra parte do conteúdo desse ficheiro.

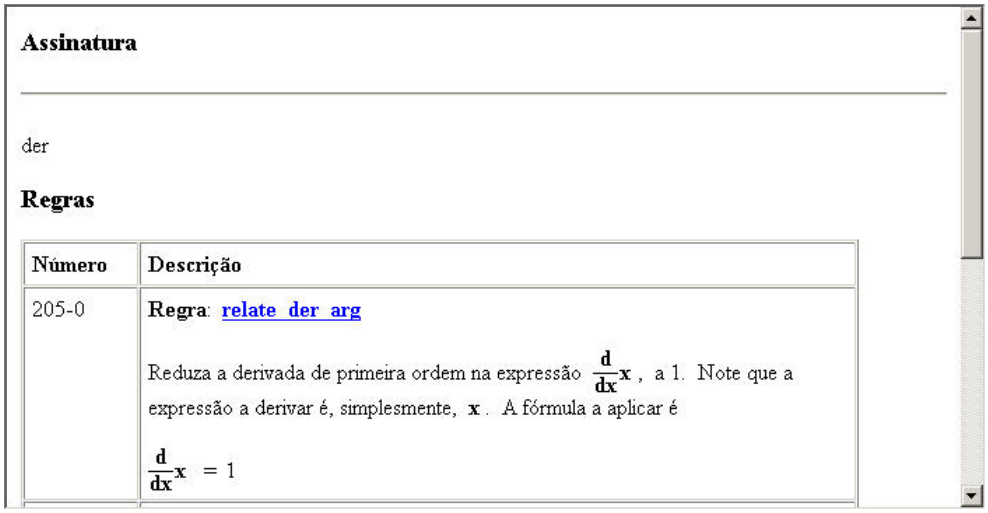



**Figura 5.4.3** ***Lista de regras com uma assinatura*** *der*

O utilizador decide aplicar a regra de decomposição de derivadas, *decompose_der*, como sendo a regra mais adequada para esse tipo de expressão. O lado esquerdo da fórmula é, neste caso, $\frac{d^k}{dx^k} f$, o que coincide, precisamente, com o contexto $\frac{d^k}{dx^k} x^k$, associado com essa expressão.

Para aplicar a regra, o utilizador necessita apenas de enviar ao Assistente a frase "*aplicar a regra decompose_der a der 1*". O Assistente aplica a regra (*decompose_der*) à primeira (*1*) componente derivada (*der*) e adiciona o resultado ao ficheiro *pratica_1.xml*. A aplicação de uma regra implica sempre a execução dos predicados *simplify*/11 e *apply_relation*/7 (Secção 4.3.1). A aplicação dessa regra é identificada como o primeiro passo dessa resolução. O Assistente informa também que a prática está ainda por resolver. A Figura 5.4.4 mostra o conteúdo desse passo, no ficheiro *pratica_1.xml*.

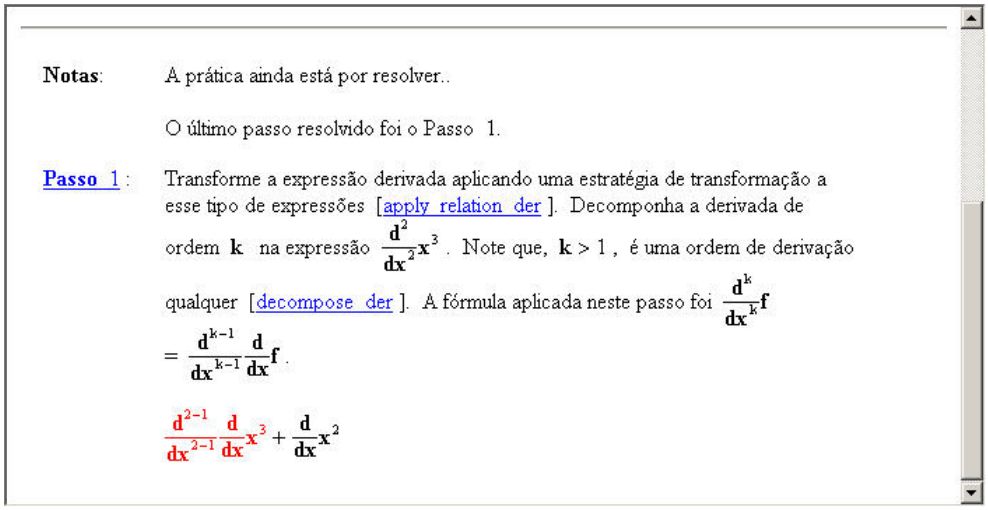



**Figura 5.4.4** ***Aplicação de uma regra*** *decompose_der* ***(Passo 1)***

O resultado desse passo, $\frac{d^{2-1}}{dx^{2-1}}\frac{d}{dx}x^3 + \frac{d}{dx}x^2$, é uma expressão equivalente que necessita de ser resolvida no contexto dessa prática. O utilizador nota que nessa expressão estão presentes duas expressões numéricas, ainda por resolver. Para as resolver, o utilizador decide aplicar uma regra de cálculo. Ele sabe que as regras de cálculo são identificadas pelo prefixo *evaluate* e que, neste caso, a regra deve possuir uma assinatura *diff* , ou seja, ser identificada como *evaluate_diff* . O utilizador decide aplicar essa regra à primeira dessas expressões. Ele envia ao Assistente a frase "*aplicar a regra evaluate_diff a diff 1*". Ele nota, porém, que com a aplicação dessa regra ambas foram resolvidas. Ele fica a saber que essas duas expressões estão intimamente ligadas uma à outra, ou seja, que a resolução duma deve conduzir imediatamente à resolução da outra. Isto é, que as expressões fazem parte da notação do mesmo operador, neste caso, a derivada. Na realidade, a regra só foi aplicada uma única vez. O utilizador fica a saber que uma expressão desse tipo não pode ser reescrita como $\frac{d}{dx^{2-1}}\frac{d}{dx}x^3 + \frac{d}{dx}x^2$. Isto é, que os expoentes que aparecem nas derivadas não representam potências mas sim a ordem de derivação dessas derivadas e que esses expoentes fazem parte da notação matemática utilizada

para representar derivadas de ordem superior. A Figura 5.4.5 mostra esse passo da resolução.

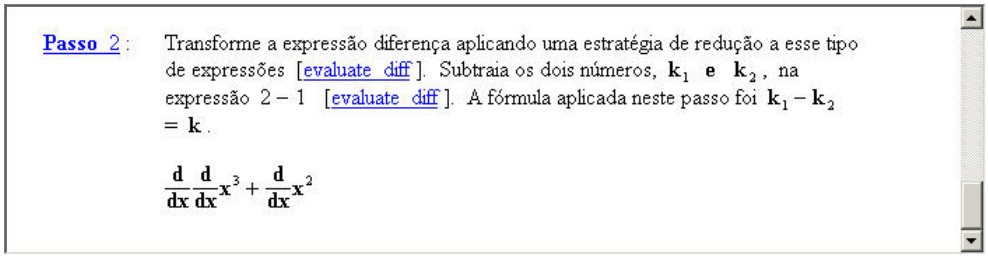



**Figura 5.4.5** ***Aplicação de uma regra*** *evaluate_diff* ***(Passo 2)***

A expressão obtida nesse passo, $\frac{d}{dx}\frac{d}{dx}x^3+\frac{d}{dx}x^2$, possui várias derivadas. O utilizador decide aplicar uma regra de derivação à expressão, $\frac{d}{dx}x^3$. Neste caso, ele opta por uma regra do tipo *relate* com uma assinatura *der_pwr*. A derivada que escolheu é a segunda nessa expressão. O utilizador sabe que essa expressão deve ser lida como "*a soma da primeira derivada da primeira derivada do cubo de x com a derivada do quadrado de x*" ou como "*sum der 1 der 1 pwr 3 x der 1 pwr 2 x*". O utilizador envia ao Assistente a frase "*aplicar a regra relate_der_pwr a der 2*". O Assistente aplica essa regra à expressão $\frac{d}{dx}x^3$ e reescreve a expressão $\frac{d}{dx}\frac{d}{dx}x^3+\frac{d}{dx}x^2$. A Figura 5.4.6 mostra esse passo da resolução. O utilizador verifica que a derivada que foi resolvida é, de facto, a segunda. O Assistente assinala essa parte da expressão a cores diferentes. Como já foi referido na Secção 2.3, a leitura operacional de uma expressão é feita em profundidade (*depth-first*), da esquerda para a direita (*left-right*).

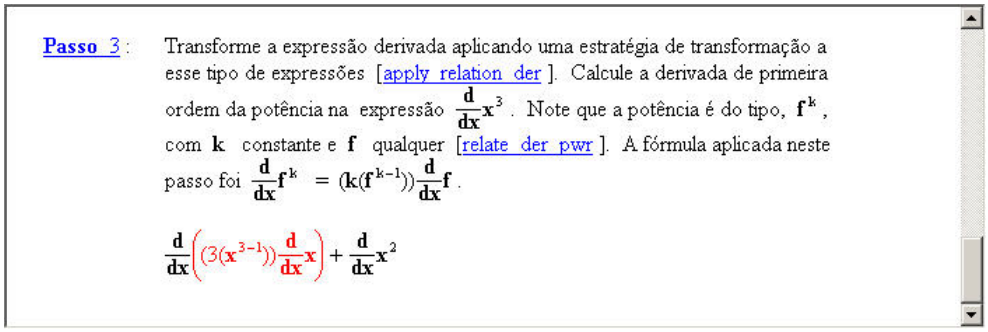



**Figura 5.4.6** ***Aplicação de uma regra*** *relate_der_pwr* ***(Passo 3)***

O resultado deste passo é uma expressão com o mesmo número de derivadas, mas onde a segunda do passo anterior foi substituída por uma mais simples, $\frac{d}{dx}x$ . O utilizador nota que a derivada, $\frac{d}{dx}x^2$ , é semelhante àquela que acabou de resolver. Ele decide, portanto, aplicar a mesma regra a essa derivada. Mas a derivada, neste caso, é a terceira nessa expressão. O utilizador envia ao Assistente a frase "*aplicar a regra relate_der_pwr a der 3*". A frase é semelhante à anterior mas com uma expressão alvo diferente. O Assistente aplica essa regra à expressão alvo (terceira derivada) e adiciona o novo passo à *pratica_1*. A Figura 5.4.7 mostra esse passo.

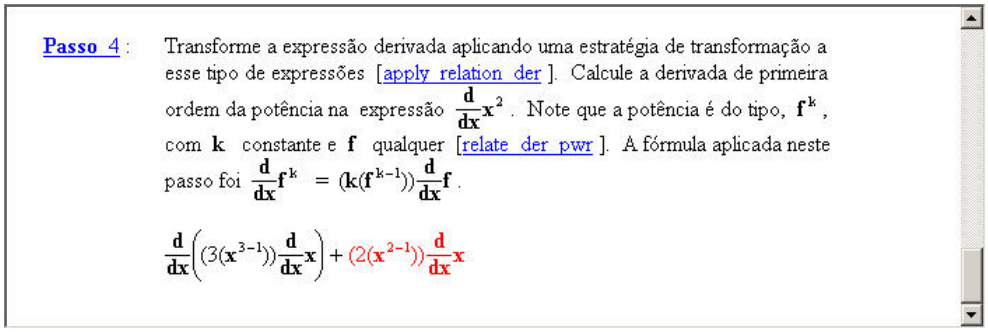



**Figura 5.4.7** ***Aplicação de uma regra*** *relate_der_pwr* ***(Passo 4)***

A expressão que é obtida neste caso contém duas expressões numéricas. Como estratégia, o utilizador sabe que as expressões numéricas devem ser as

primeiras a ser resolvidas. Ele opta pela primeira dessas duas expressões. A escolha neste caso não tem qualquer influência estratégica para essa resolução. O utilizador envia então ao Assistente a frase "*aplicar a regra evaluate_diff a diff 1*". A expressão alvo, neste caso, é a primeira pois numa leitura operacional dessa expressão, essa seria a expressão potência lida primeiro. O utilizador, nesta altura, já consegue "*ver*" com mais facilidade essa ordem de leitura ou precedência. O Assistente aplica essa regra à primeira dessas duas expressões e reescreve a expressão como o passo seguinte dessa resolução. A Figura 5.4.8 mostra esse passo. O utilizador nota que neste caso apenas aquela que escolheu como alvo foi resolvida. Ele reconhece que, neste caso, as duas expressões não estão relacionadas por qualquer tipo de notação e que devem, portanto, ser resolvidas separadamente.

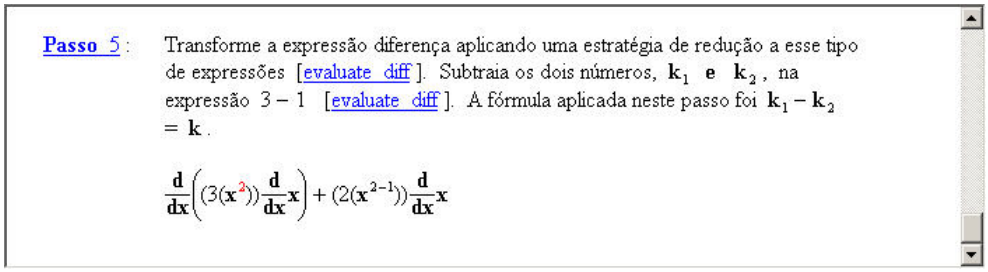



**Figura 5.4.8** ***Aplicação de uma regra*** *evaluate_diff* ***(Passo 5)***

O utilizador nota também que a outra expressão pode ser resolvida aplicando exactamente a mesma regra. Ele envia de novo ao Assistente a frase "*aplicar a regra evaluate_diff a diff 1*". A expressão alvo passou a ser a primeira e a única expressão desse tipo na expressão obtida do passo anterior. O Assistente aplica essa regra à expressão alvo e adiciona esse passo à *prática_1*. A Figura 5.4.9 mostra esse passo.

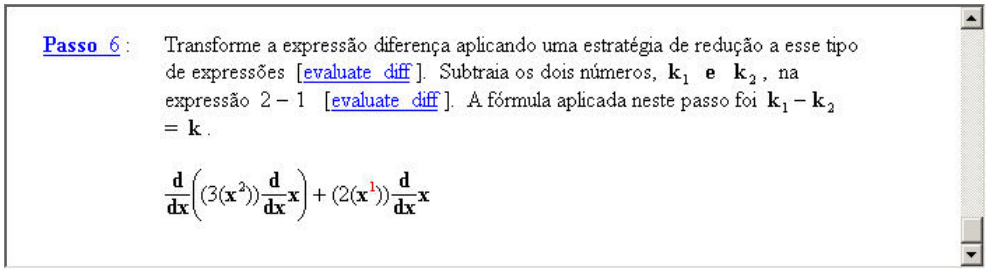



**Figura 5.4.9** ***Aplicação de uma regra*** *evaluate_diff* ***(Passo 6)***

O resultado obtido possui ainda várias derivadas por resolver, mas mais simples. O utilizador nota também que a segunda potência, $x^1$, pode ser simplificada. Ele sabe que essa expressão é uma potência onde um dos argumentos possui uma característica especial. Trata-se de uma relação especial (*relate*) entre o operador (*pwr*) e um dos seus argumentos (*arg*). Essa relação possui uma assinatura *pwr*. O utilizador envia ao Assistente a frase "*aplicar a regra relate_pwr_arg a pwr 2*". Neste caso, a regra envolve apenas um dos argumentos (*arg*) dessa potência. Se a relação envolvesse ambos os argumentos, o sufixo seria *args* (Secção 4.3.1). O Assistente aplica essa regra à segunda potência, nessa expressão, e adiciona esse passo à resolução dessa prática. A Figura 5.4.10 mostra esse passo.

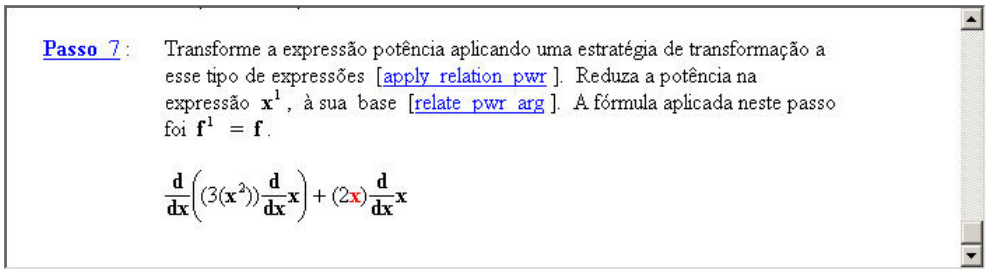



**Figura 5.4.10** ***Aplicação de uma regra*** *relate_pwr_arg* ***(Passo 7)***

O utilizador nota que existem ainda três derivadas por resolver. Por leitura, ele sabe que a primeira é uma derivada mais complicada que qualquer uma das

outras. Ele decide, portanto, resolver primeiro as duas que lhe parecem ser as mais simples. Ele opta pela primeira dessas duas derivadas, $\frac{d}{dx}x$. Essa derivada é precisamente a segunda nessa expressão. O utilizador sabe que assim é porque, numa leitura operacional, essa é a expressão derivada que é lida em segundo lugar, O utilizador envia ao Assistente a frase "*aplicar a regra relate_der_arg a der 2*". O Assistente aplica essa regra à segunda derivada nessa expressão e adiciona esse passo à prática_1. A Figura 5.4.11 mostra esse passo.

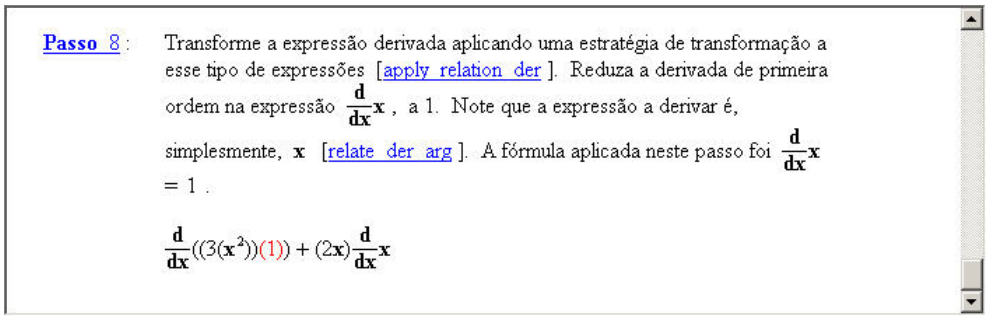



**Figura 5.4.11** ***Aplicação de uma regra*** *relate_der_arg* ***(Passo 8)***

Para resolver a outra derivada, o utilizador envia ao Assistente a mesma frase, ou seja, a frase "*aplicar a regra relate_der_arg a der 2*". Com a resolução da outra, esta passou a ser a segunda derivada nessa expressão. O Assistente aplica essa regra à derivada alvo e adiciona esse passo à resolução dessa prática. A Figura 5.4.12 mostra esse passo.

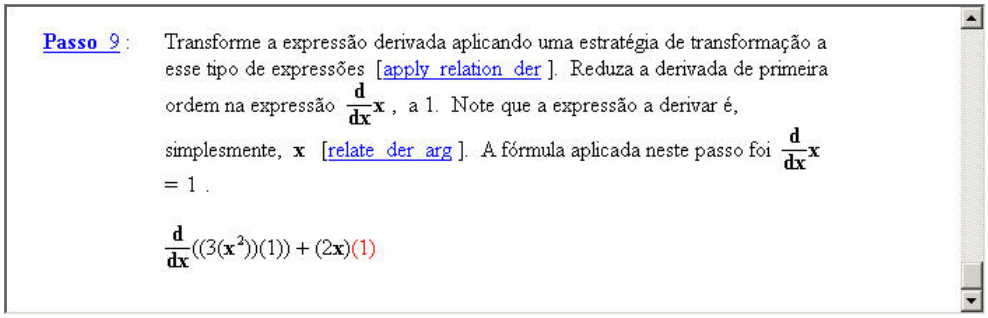



**Figura 5.4.12** ***Aplicação de uma regra*** *relate_der_arg* ***(Passo 9)***

O resultado dessa aplicação é uma expressão com vários produtos. Dois deles (os produtos por 1) podem ser ainda reduzidos. O utilizador decide aplicar, neste caso, uma regra do tipo *relate*. Ele opta por reduzir o primeiro desses dois produtos, ou seja, a expressão $(\ldots)(1)$. Esse produto é, de facto, o primeiro nessa expressão. Numa leitura operacional, o produto, $3x^2$ é o segundo, o outro produto $(\ldots)(1)$, o terceiro. A presença dos parênteses é essencial para uma leitura operacional dessa expressão. A regra aqui envolve apenas um dos argumentos e, portanto, é identificada como *relate_prod_arg*. O utilizador envia então ao Assistente a frase "*aplicar a regra relate_prod_arg a prod 1*". O Assistente aplica essa regra e adiciona esse passo à resolução. A Figura 5.4.13 mostra esse passo.

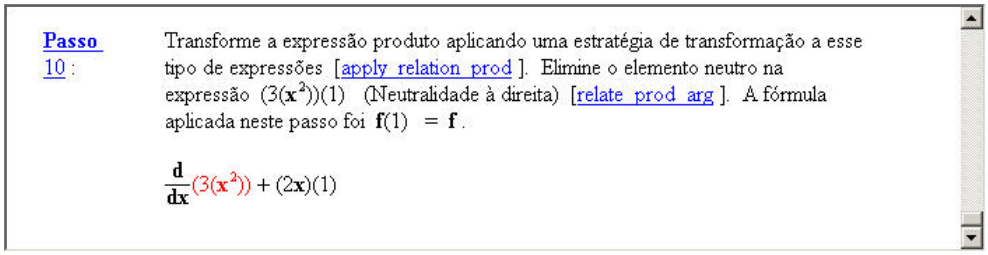



**Figura 5.4.13** ***Aplicação de uma regra*** *relate_prod_arg* ***(Passo 10)***

O segundo desses produtos, $(2x)(1)$, aparece agora como o segundo. Nessa expressão, o argumento da derivada, $3x^2$, é o primeiro. Como já referimos atrás, a leitura é feita em profundidade e da esquerda para a direita. O utilizador envia ao

Assistente a frase "*aplicar a regra relate_prod_arg a prod 2*". O Assistente aplica essa regra a esse produto e adiciona o passo a essa prática. A Figura 5.4.14 mostra esse passo.

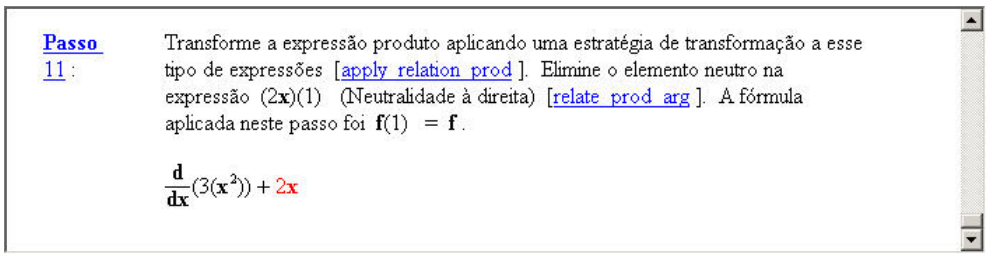



**Figura 5.4.14** ***Aplicação de uma regra*** *relate_prod_arg* ***(Passo 11)***

O utilizador nota que existe ainda uma derivada por resolver. Essa expressão possui uma assinatura *der_prod* . Ele sabe isso porque numa leitura operacional dessa expressão, o produto é o único operador que faz parte dos argumentos dessa derivada. Como se sabe, a assinatura de uma expressão é determinada apenas pela estrutura triangular da sua componente principal (Secção 2.4). Como o utilizador já sabe como resolver este tipo de derivadas, ele decide testar as suas capacidades e fornecer o resultado do passo seguinte. O utilizador envia ao Assistente a frase "*verificar a expressão sum prod 3 der 1 pwr 2 x prod 2 x*", para que o Assistente possa verificar se a expressão $3\frac{\mathrm{d}}{\mathrm{dx}}x^2+2x$, é ou não equivalente.

O Assistente verifica se essa expressão é equivalente, resolvendo, ele próprio, essa expressão, de todas as maneiras possíveis. A verificação de um resultado não afecta a sua memória colectiva (Secção 4.4.2) da resolução. Essa tarefa pode não ser fácil para expressões mais complexas. A expressão é considerada equivalente se fizer parte de uma das cadeias de resolução obtidas pelo Assistente. A expressão não tem de derivar, necessariamente, de nenhum passo anterior, ou de satisfazer a qualquer ordem cronológica de apresentação. A

expressão tem de fazer parte apenas de uma das cadeias de derivação do Assistente. Como resultado, o Assistente regista apenas esse passo como tendo sido introduzido manualmente. A Figura 5.4.15 mostra esse passo.

Não é possível, neste caso, indicar qual foi a estratégia ou regra, aplicadas. Nesta versão do Assistente, não é possível estabelecer uma relação de equivalência entre esse passo e qualquer outro nessa cadeia de derivação. Isso só seria possível resolvendo o passo anterior e comparando a seguir cada um dos passos dessa resolução com a expressão que se pretende verificar. De facto, o resultado obtido podia ter sido derivado aplicando várias estratégias e regras de reescrita, em simultâneo. Não é também possível verificar todas as formas equivalentes de uma expressão desta forma. Para o conseguir, seria necessário também resolver a expressão que se pretende verificar.

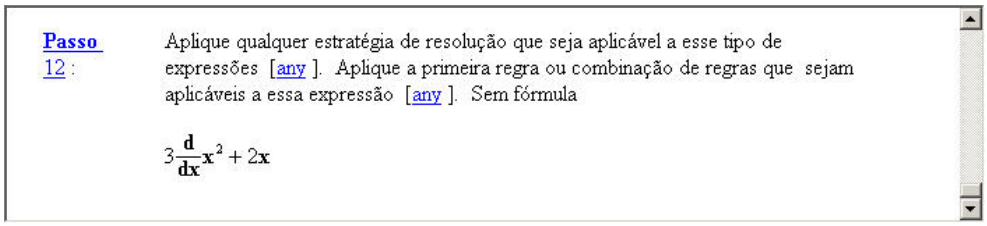



**Figura 5.4.15** *Verificação de um resultado (Passo 12)*

O resultado introduzido, neste caso, contém ainda uma derivada por resolver. O utilizador decide, mais uma vez, fazê-lo manualmente. Ele envia ao Assistente a frase "*verificar a expressão sum prod 3 prod 2 x prod 2 x*", para que o Assistente verifique a expressão $3(2x)+2x$. O Assistente verifica que a expressão é correcta e adiciona esse passo como manual. A Figura 5.4.16 mostra esse passo.

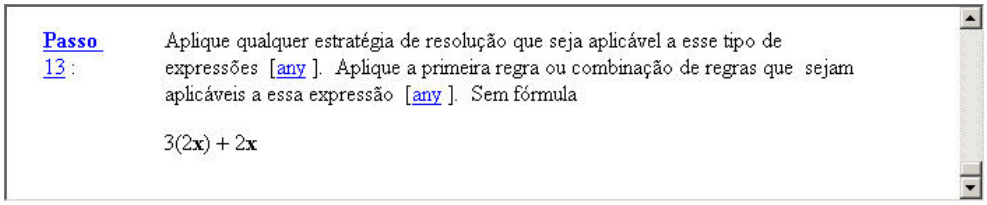



**Figura 5.4.16** *Verificação de um resultado (Passo 13)*

O utilizador nota que nessa expressão existe um termo, $2x$, comum às duas parcelas e decide pô-lo em evidência. Ele não sabe como fazê-lo, mas sabe que a expressão possui uma assinatura *sum_prod* . Ele envia ao Assistente a frase "*listar a assinatura sum_prod*". O Assistente envia para o *browser* uma lista de todas as regras com esse tipo de assinatura. A Figura 5.4.17 mostra essa lista.

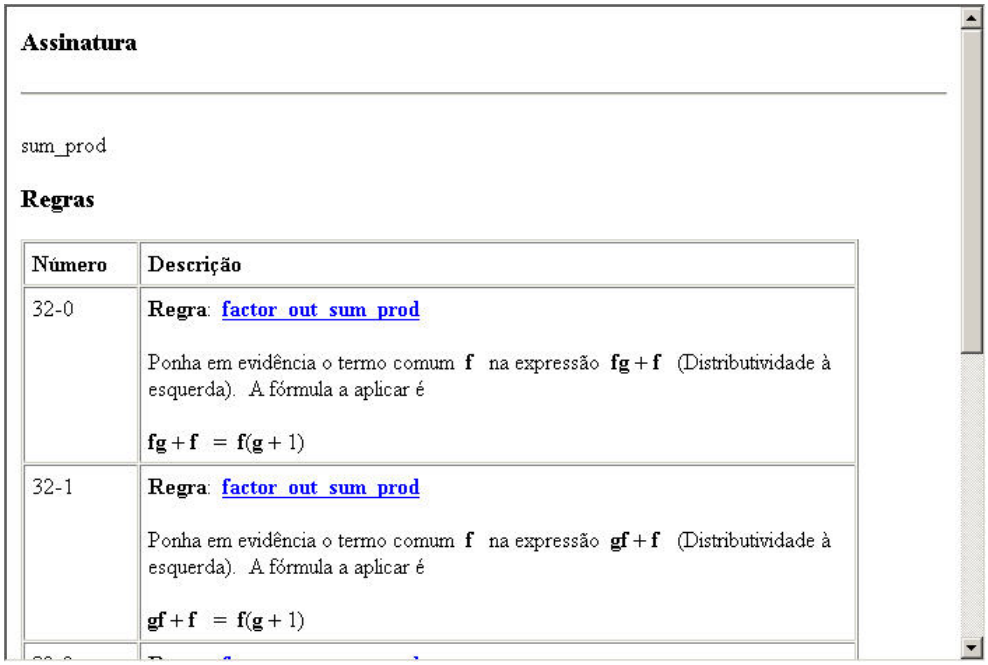



**Figura 5.4.17** *Lista de regras com a assinatura* sum_prod

O utilizador consulta a lista e decide aplicar a regra *factor_out_sum_prod* à expressão soma. Ele envia ao Assistente a frase "*aplicar a regra factor_out_sum_prod a prod 1*". O Assistente aplica essa regra à expressão soma e

adiciona esse passo à *pratica_1*. A Figura 5.4.18 mostra esse passo.

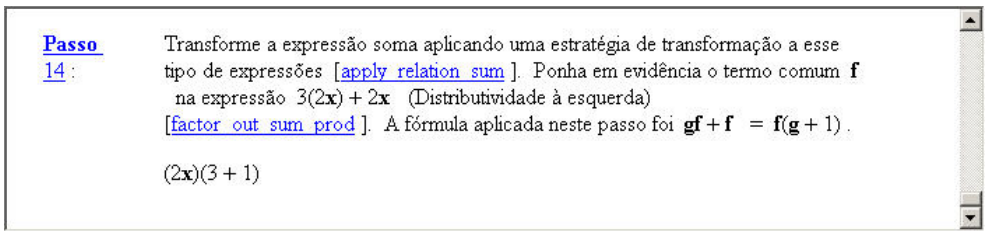



**Figura 5.4.18** ***Aplicação de uma regra*** *factor_out_sum_prod* ***(Passo 14)***

O utilizador decide concluir essa prática e, mais uma vez, introduzir o resultado manualmente. Ele envia ao Assistente a frase "*verificar a expressão prod 8 x*". O Assistente verifica a expressão e adiciona esse passo como manual. A Figura 5.4.19 mostra esse passo.

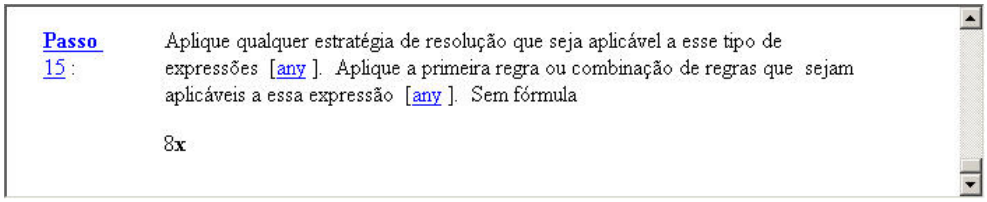



**Figura 5.4.19** ***Verificação de um resultado (Passo 15)***

Essa expressão parece estar na sua forma normal. Como sabemos, uma expressão na forma normal não é redutível e, portanto, deve representar o fim de qualquer cadeia de derivação. Para verificar que assim é, o utilizador envia ao Assistente a frase "*resolver o passo seguinte*". O Assistente tenta resolver essa expressão mas não consegue, pois não existem regras de reescrita com essa assinatura ( *prod* ) e para esse contexto ( *kx* ). A prática está, portanto, resolvida e o resultado é $8x$ .

# Capítulo 6 Conclusões

## AS CONCLUSÕES DA TESE E O TRABALHO FUTURO

## 6.1 Conclusões

O objecto de estudo desta tese é precisamente a resolução simbólica de expressões matemáticas, no contexto do ensino da Matemática, ao nível do secundário. A resolução simbólica de expressões é um método de resolução que visa, essencialmente, a obtenção da forma mais simples dessas expressões, por reescrita axiomática.

Com esta tese, pretendemos demonstrar que

***Hipótese I:***

> *É possível resolver, de forma simbólica e por reescrita axiomática, qualquer expressão matemática que possa ser completamente caracterizada pela estrutura triangular da sua componente principal e que seja simplificável.*

De facto, todas as expressões, incluídas nesta tese, possuem uma estrutura triangular e são completamente caracterizadas pela estrutura triangular da sua componente principal (Secção 2.4). Todas essas expressões podem ser decompostas num número finito de estruturas triangulares básicas e transformadas,

individualmente, por reescrita axiomática.

A decomposição de uma estrutura triangular preserva a estrutura triangular de todas as suas componentes e vice-versa. A reescrita axiomática é efectuada apenas com base na estrutura triangular de cada componente. A reescrita axiomática está intimamente relacionada com o conceito de estrutura triangular. A estrutura triangular de uma componente é uma das cinco (5) estruturas triangulares básicas definidas nesta tese (Secção 2.4).

Uma reescrita axiomática produz apenas estruturas triangulares básicas. As estruturas triangulares podem portanto ser comparadas com base no seu grau de complexidade (Secção 2.4.2), ou seja, com base na estrutura da sua componente principal. A reescrita axiomática é feita de uma forma contextualizada, ou não, mas apenas ao nível de cada componente. O contexto em que uma componente está inserida ou os argumentos em que ela serve de contexto não são afectados pela reescrita dessa componente. A contextualização de uma reescrita não afecta a reescrita. A reescrita é feita por composição lógica. A composição lógica de uma reescrita com o resto da estrutura depende dos contextos em que está inserida.

Uma expressão é simplificável se, pelo menos, uma das suas componentes for um *redex*. Um *redex* é uma expressão redutível, por reescrita axiomática, ou seja, uma expressão para a qual existe, pelo menos, uma regra de reescrita com essa assinatura capaz de a reduzir ou converter (Secção 2.5). O resultado de uma redução é sempre uma estrutura mais simples. O resultado de uma conversão pode ser uma expressão mais simples ou uma expressão mais complexa mas potencialmente redutora. Uma expressão potencialmente redutora agrava o grau de complexidade de uma estrutura mas compensa esse agravamento com a formação de novos *redexes*. Note que o agravamento do grau de complexidade de uma estrutura não pode ir além de 1 nível estrutural ( $N0 \rightarrow N1$ , $N1 \rightarrow N2$ ou $N2 \rightarrow N3$ ), ou seja, a adição de uma componente. Com a formação de novos *redexes*, em número superior

ao que existia, o resultado líquido de uma reescrita é sempre uma estrutura mais simples.

Podemos, portanto, concluir que se uma expressão possuir uma estrutura triangular, ela pode ser decomposta num número finito de estruturas triangulares básicas e resolvida, simbolicamente, por reescrita axiomática, se pelo menos uma dessas componentes for um *redex*. O resultado de reescrita é uma expressão mais simples ou uma expressão que se pode tornar mais num número finito de passos.

O conceito de estrutura triangular foi motivado pelas estruturas exibidas pelos lados esquerdos das regras matemáticas utilizadas na base axiomática. Expressões com essa estrutura possuem um significado operacional, ou computacional, bem definido. A estrutura triangular tornou possível uma leitura totalmente inequívoca dessas expressões e a sua caracterização com base na estrutura triangular da sua componente principal. Com esse tipo de estrutura foi possível classificar as regras de reescrita e organizar a base axiomática por classes e dispor as regras de reescrita por ordem decrescente do seu poder redutor.

Com esta tese, pretendemos demonstrar também que

***Hipótese II:***

> *A resolução simbólica, por reescrita axiomática, de expressões matemáticas com estruturas triangulares, é convergente se a base axiomática for completa.*

A resolução simbólica de expressões é um sistema de reescrita, baseado na transformação lógica de expressões, por reescrita axiomática. Um sistema de reescrita é convergente, se for confluente e terminante (Secção 3.5).

A terminação de uma resolução é garantida pelo princípio de exclusão e pela finitude da base axiomática. De facto, o princípio de exclusão garante a

unicidade de qualquer expressão obtida por reescrita axiomática. A finitude da base axiomática garante que só um número finito de regras pode ser aplicado. As formas equivalentes de uma expressão só podem interceptar a sua memória, uma única vez. Isto é, na memória não podem existir formas duplicadas dessas expressões. A memória colectiva de uma resolução é, por definição, o conjunto ordenado de todas as reescritas axiomáticas efectuadas sobre a expressão que é objecto dessa resolução (Secção 4.4.2). Isso inclui, naturalmente, a forma mais simples dessa expressão. De facto, o fecho transitivo de uma reescrita axiomática é sempre a forma mais simples da expressão. Isso é garantido pelo facto da base axiomática conter apenas regras redutoras ou potencialmente redutoras.

A confluência de uma resolução simbólica é garantida pela memória colectiva dessa resolução e pela completude da sua base axiomática. Pelo princípio de exclusão, uma solução só pode interceptar uma outra, uma única vez. Porém, qualquer solução pode ser interceptada mais do que uma vez. Isto é, a intercepção entre duas soluções, quaisquer, é um conjunto singular (*singleton*). A forma interceptada é, precisamente, o resultado da reescrita para a qual não existem estratégias alternativas para a sua resolução. Uma vez reescrita, a expressão só pode ser obtida a partir da sua memória. O que significa que uma solução que intersecte uma outra pode ser completada a partir da sua memória.

Como, por definição, uma solução é um conjunto ordenado de transformações lógicas obtidas por reescrita axiomática, todas as soluções possíveis de uma resolução simbólica devem fazer parte da sua memória colectiva. Se a base axiomática for completa, então qualquer solução deve conduzir à forma mais simples da expressão. De facto, a completude de uma base axiomática garante a irredutibilidade de uma reescrita axiomática. Assim, se existir mais do que uma solução, as soluções devem interceptar uma outra, pelo menos, uma vez. De facto, se tal não acontecesse, existiriam na memória soluções que não conduziriam à forma mais simples, o que seria uma contradição. Podemos, portanto, dizer que

todas as soluções obtidas por resolução simbólica convergem para a forma mais simples da expressão, se a base axiomática for completa. De facto, a forma mais simples é aquela para a qual todas as intercepções convergem. Podemos também dizer que, se a base axiomática for completa, o resultado de uma resolução não depende da estratégia utilizada.

Podemos, portanto, concluir que, se a base axiomática for completa, a resolução simbólica de expressões por reescrita axiomática é convergente. A priori, é difícil garantir a completude de uma base axiomática. Mas, como já foi referido, esta tese tenta tratar essencialmente expressões que são lidadas ao nível do secundário. Assim, para estas, pelo menos, a base parece ser relativamente completa. Essa base está listada no Apêndice B. O Apêndice C apresenta um conjunto de exemplos que pretende ser representativo da generalidade dos programas do secundário. Para todos os exemplos, os testes efectuados demonstram a completude da base axiomática apresentada nesta tese.

O desenvolvimento de uma linguagem computacional, com as características que defendemos nesta tese, poderá vir a contribuir, no futuro, para o desenvolvimento de aplicações para o ensino e aprendizagem da Matemática, capazes de actuar como tutores, ou assistentes electrónicos de Matemática, e de comunicar de uma forma natural.

Um dos contributos desta tese foi o desenvolvimento da estrutura triangular como uma forma computável, e bastante eficiente, de se representar expressões matemáticas, numa lógica de 1ª ordem. Como aplicação prática, desenvolvemos um programa Prolog, capaz de resolver simbolicamente expressões matemáticas, passo a passo, de forma automática.

## 6.2 O Trabalho Futuro

Pensamos no futuro vir a abordar o conceito de equação/inequação como uma forma de expressão lógica, e incluir operadores matemáticos como o integral e o limite. Consideramos que as equações/inequações podem ser tratadas também, de forma axiomática.

A axiomatização de artifícios é uma área que pensamos ser grande utilidade para a resolução simbólica de expressões por reescrita axiomática. A aplicação de artifícios como, por exemplo, a aplicação da regra do operador inverso a ambos os membros de uma equação/inequação, é uma regra que Bundy refere em [BUN85] como sendo uma regra não documentada mas bastante útil. O conceito de resolução simbólica, por reescrita axiomática, parece ser aplicável também a equações e inequações.

O cálculo de limites e o cálculo integral parecem ser duas outras áreas onde esse tipo de tratamento possa vir a ser útil. Tal como a conversão de operadores (potências e raízes) e a conversão de bases (logaritmos e exponenciais), a aplicação de artifícios deve ser tratado também de forma contextualizada.

A utilização da notação matemática na escrita de expressões é uma outra área que interessa investigar no futuro. A estrutura triangular poderá a ser um contributo válido para esse tipo de investigação. Os progressos alcançados na área do reconhecimento da escrita manuscrita de expressões matemáticas [CHA00, CHA01] sugerem novas possibilidades para a escrita de expressões matemáticas. Essa nova capacidade de escrita irá permitir que frases como "*definir a expressão* $\log_e x^2$" possam ser manipuladas graficamente. A edição e a selecção de componentes poderão ser feitas directamente a partir das respectivas representações gráficas. A selecção de componentes é uma área de interesse para a resolução simbólica de expressões matemáticas por via axiomática, pois irá permitir uma

interacção mais estreita entre a escrita gráfica e a resolução simbólica de expressões ou reescrita axiomática. Por exemplo, a reescrita axiomática de uma expressão como $\log_e 4^2$ poderia ser feita apontando directamente para a componente que se pretende reescrever, e através de determinados "*movimentos*" ou "*gestos*", como, por exemplo, o "*drag & drop*" (arrastar e largar), proceder à reescrita dessas componentes e expressões. Por exemplo, no caso da expressão $\log_e 4^2$, arrastar ("*drag*") o expoente da potência e largá-lo ("*drop*") atrás do logaritmo teria o mesmo efeito que aplicar a regra *relate_log_pwr*, ou seja, efectuar a transformação $\log_e 4^2 \rightarrow 2\log_e 4$. Esse "movimento" teria um significado operacional bem definido. A transformação seria efectuada "*in loco*", ou seja, no mesmo local onde a componente estaria localizada. Um duplo clique sobre o argumento do logaritmo poderia ter o mesmo efeito que, por exemplo, a factorização desse argumento, ou seja, a execução da transformação $2\log_e 4 \rightarrow 2\log_e 2^2$. Esses "*movimentos*" ou "*gestos*" seriam sensíveis ao tipo de expressão para à qual apontam ou seleccionam.

Todos esses "*movimentos*" ou "*gestos*" têm em comum o aspecto atómico dessas operações. A reescrita axiomática de expressões matemáticas é baseada na atomicidade dessas operações. Essa atomicidade, no caso da resolução simbólica de expressões, por reescrita axiomática, é garantida pela estrutura triangular dessas expressões.

# Apêndice A O PROGRAMA PROLOG

## A.1 Reescrita de Expressões

Nesta secção está listado uma amostra do código para cada um dos tipos de reescrita abordados na Secção 4.3.1 deste documento. Os comentários estão escritos em inglês por ter sido essa a linguagem utilizada durante o seu desenvolvimento.

### A.1.1 Reescrita por Cálculo

A reescrita por cálculo é executada pelo predicado *evaluate* / 7 . O predicado é mantido no ficheiro *evaluate.pl* . Nesse ficheiro estão definidos todas as regras $\kappa$-*op* .

O Programa A.1-1 mostra a implementação Prolog de uma regra de reescrita $\kappa$-*sum* .

**Programa A.1-1** ***Regra de reescrita*** $\kappa$-*sum*

| | |
|---|---|
| *Regra:* | *LHS -> RHS* |
| *Lado esquerdo da regra:* | *sum(Arg1,Arg2)* |
| *Resultado da reescrita:* | *RHS* |
| *Posição da estrutura onde a regra é aplicada:* | *P* |
| *Profundidade a que a estrutura se encontra:* | *Lv* |
| *Estratégia aplicada:* | *St = evaluate_sum* |
| *Regra aplicada:* | *R = evaluate_sum* |
| *Contexto ou padrão da regra aplicada:* | *C = sum(k(1),k(2))* |

```
1   evaluate(sum(Arg1,Arg2),RHS,P,Lv,St,R,C) :-
```

```
2     (
3       R = evaluate_sum,
4       C = sum(k(1),k(2))
5     ),
6     (
7       both_regular_numbers(Arg1,Arg2)
8     ),
9     RHS is Arg1+Arg2.
```

A regra $\kappa$-*sum*, representada aqui, unifica as expressões $\text{sum}(k_1, k_2)$ e $\text{sum}(Arg_1, Arg_2)$, e efectua o cálculo numérico, $Arg_1 + Arg_2$ (Programa A.1-1-9). A instanciação das variáveis é feita por unificação (Secção 3.3.2). O unificador mais geral (Secção 3.3.2) é, neste caso, a substituição $\sigma = \{Arg_1 \mapsto k_1, Arg_2 \mapsto k_2\}$. As variáveis, $Arg_i$, são as variáveis livres na transformação lógica , $C \mid L \to \lambda_\kappa(R)$. A fórmula de cálculo é, neste caso,

$$Arg_1 + Arg_2 \to RHS \tag{1}$$

A resolvente (Secção 3.3.2) da transformação, $\lambda_\kappa(R)\sigma$, é simplesmente o resultado da aplicação[60], $([Arg_1, Arg_2]\lambda_\kappa(R))\sigma$ [61]. O cálculo numérico, ou reescrita -$\kappa$, $\lambda_\kappa(R)$, é efectuado pelo predicado $is/2$ (Programa A.1-1-9). Neste caso, a reescrita é simplesmente a instanciação da variável $RHS$.

A regra é condicional, ou seja, apenas aplicável se um certo número de requisitos operacionais, $C_i$ (Programa A.1-1-2/8), forem satisfeitos. O requisito (Programa A.1-1-7) condiciona a sua aplicação a números inteiros apenas (*regular_numbers*). O requisito (Programa A.1-1-3) identifica a regra como sendo uma regra do tipo *evaluate_sum*, ou seja uma regra $\kappa$-*sum*. O requisito (Programa

[60] Neste caso, trata-se da aplicação de uma regra de conversão η [VAL00].

[61] A expressão $([x]\alpha)a$ representa a aplicação de uma abstracção $[x]\alpha$ a um padrão $a$. Segundo [VAL00], a aplicação tem uma interpretação computacional que pode ser expressa pela transformação $\alpha[a/x]$, ou seja, pela aplicação de uma substituição de variáveis por termos.

A.1-1-4) identifica o padrão do lado esquerdo da fórmula como sendo do tipo $k_1 + k_2$.

## A.1.2 Reescrita por Redução ou Conversão axiomática

As reescritas por redução ($\tau$-*op*) ou conversão axiomática ($\alpha,\beta,\gamma,\psi$-*op_op* ou $\alpha,\beta,\gamma,\psi$-*op_op_op*) são executadas pelo predicado *apply_relation* / 7.

O Programa A.1-2 mostra a implementação Prolog de uma regra de reescrita $\tau$-*sum*. As regras para o operador *sum* estão mantidas no ficheiro *apply_relation_sum.pl*.

**Programa A.1-2** ***Regra de reescrita*** $\tau$-*sum*

```
Regra:                          LHS -> RHS
Lado esquerdo da regra:         sum(Arg1,Arg2)
Resultado da reescrita:         RHS
Posição da estrutura onde
        a regra é aplicada:     P
Profundidade a que a
        estrutura se encontra:  Lv
Estratégia aplicada:            St = apply_relation_sum
Regra aplicada:                 R  = relate_sum_arg
Contexto ou padrão da
        regra aplicada:         C  = sum(f,0)
```

```
1   apply_relation(sum(Arg1,Arg2),Arg1,P,Lv,St,R,C):-
2     (
3       R = relate_sum_arg,
4       C = sum(f,0)
5     ),
6     (
7       except_number(Arg1)
8     ),
9     (
10      zero(Arg2)
11    ).
```

A regra $\tau$-*sum*, representada aqui, unifica as expressões sum($f$,0) e, sum($Arg_1, Arg_2$). O unificador mais geral (Secção 3.3.2), neste caso, é simplesmente a substituição $\sigma = \{Arg_1 \mapsto f, Arg_2 \mapsto 0\}$. As variáveis $Arg_i$ são as variáveis livres na transformação condicional, $C \mid L \to \lambda_\tau(R)$, ou seja, as variáveis livres na fórmula

$$Arg_1 + Arg_2 \to Arg_1 \qquad (1)$$

A resolvente (Secção 3.3.2) da reescrita $\tau$-*sum*, $\lambda_\tau(R)\sigma$, é simplesmente o resultado da aplicação $([Arg_1, Arg_2]\lambda_\tau(R))\sigma$ ao lado direito $\lambda_\tau(R_L)$ da fórmula (1), ou seja, a instanciação da variável $Arg_1$.

O Programa A.1-3 mostra a implementação Prolog de uma regra de reescrita $\alpha$-*log_pwr*. As regras para o operador *log* estão mantidas no ficheiro *apply_relation_log.pl*.

**Programa A.1-3** ***Regra de reescrita*** $\alpha$*-log_pwr*

| | |
|---|---|
| *Regra:* | *LHS -> RHS* |
| *Lado esquerdo da regra:* | *log(Arg1,pwr(Arg21,Arg22))* |
| *Resultado da reescrita:* | *RHS* |
| *Posição da estrutura onde a regra é aplicada:* | *P* |
| *Profundidade a que a estrutura se encontra:* | *Lv* |
| *Estratégia aplicada:* | *St = apply_relation_log* |
| *Regra aplicada:* | *R = relate_log_pwr* |
| *Contexto ou padrão da regra aplicada:* | *C = log(a,pwr(k(1),k(2)))* |

```
1  apply_relation(log(Arg1,pwr(Arg21,Arg22)),RHS,P,Lv,St,R,C):-
2    (
3      R = relate_log_pwr,
4      C = log(a,pwr(k(1),k(2)))
5    ),
```

```
 6    (
 7      any_base(Arg1)
 8    ),
 9    (
10      regular_number(Arg21);
11      (constant(Arg21), div(Arg21), except(reducible(Arg21)))
12    ),
13    (
14      constant(Arg22)
15    ),
16    (
17      except_both_zeros_or_all_ones(Arg21,Arg22)
18    ),
19    prod(Arg21,NArg2,RHS), log(Arg1,Arg22,NArg2).
```

A regra $\alpha$-*log_pwr*, representada aqui, unifica as expressões $\log(a, \text{pwr}(k_1, k_2))$ e $\log(Arg_1, \text{pwr}(Arg_{21}, Arg_{22}))$. O unificador mais geral (Secção 3.3.2) é a substituição $\sigma = \{Arg_1 \mapsto a, Arg_{21} \mapsto k_1, Arg_{22} \mapsto k_2\}$. As variáveis $Arg_1$ e $Arg_{2j}$ são as variáveis livres na transformação condicional $C \mid L \to \lambda_\alpha(R)$, ou seja, as variáveis livres na fórmula

$$\log_{Arg_1}(Arg_{22})^{Arg_{21}} \to Arg_{21} \log_{Arg_1} Arg_{22} \qquad (1)$$

A resolvente (Secção 3.3.2) da reescrita $\alpha$-*log_pwr*, $\lambda_\alpha(R)\sigma$, é simplesmente o resultado da aplicação $([Arg_1, Arg_{21}, Arg_{22}]\lambda_\alpha(R_L))\sigma$ ao lado direito $\lambda_\alpha(R)$ da fórmula (1). Uma reescrita, $\alpha$-*log_pwr*, envolve a composição lógica de construtores $\lambda$-*prod* e $\lambda$-*log* (Programa A.1-3-19).

O Programa A.1-4 mostra a implementação Prolog de uma regra de reescrita $\alpha$-*sum_sum*. As regras para o operador *sum* estão mantidas no ficheiro *apply_relation_sum.pl* .

**Programa A.1-4** ***Regra de reescrita*** $\alpha$-*sum_sum*

```
Regra:                          LHS -> RHS
Lado esquerdo da regra:         sum(sum(Arg11,Arg12),Arg2)
Resultado da reescrita:         RHS
Posição da estrutura onde
        a regra é aplicada:     P
Profundidade a que a
        estrutura se encontra:  Lv
Estratégia aplicada:            St = apply_relation_sum
Regra aplicada:                 R  = associate_sum_sum
Contexto ou padrão da
        regra aplicada:         C  = sum(sum(f,g(1)),g(2))
```

```
 1  apply_relation(sum(sum(Arg11,Arg12),Arg2),RHS,P,Lv,St,R,C):-
 2    (
 3      R = associate_sum_sum,
 4      C = sum(sum(f,g(1)),g(2))
 5    ),
 6    (
 7      except_number(Arg11)
 8    ),
 9    (
10      except_both(sum(Arg2),diff(Arg2))
11    ),
12    (
13      except_both_zeros(Arg12,Arg2)
14    ),
15    (
16      akin(sum,Arg12,Arg2)
17    ),
18    sum(Arg11,NArg2,RHS), sum(Arg12,Arg2,NArg2),
19    (
20      reducible(NArg2)
21    ).
```

A regra $\alpha$-*sum_sum* (Programa A.1-4-01), representada aqui, unifica as expressões sum(sum($f, g_1$), $g_2$) e sum(sum($Arg_{11}, Arg_{12}$), $Arg_2$) e associa os termos $Arg_{12}$ e $Arg_2$ (Programa A.1-4-18) se existir entre esses termos uma afinidade operacional no contexto de somas (Programa A.1-4-16 e 20).

O Programa A.1-5 mostra a implementação Prolog de uma regra de reescrita

*β-log_root* , com reescrita contextualizada. As regras para o operador *root* estão mantidas no ficheiro *apply_relation_root.pl* .

**Programa A.1-5** ***Regra de reescrita*** *β-log_root*

| | |
|---|---|
| *Regra:* | *LHS -> RHS* |
| *Lado esquerdo da regra:* | *log(Arg1,root(Arg21,Arg22))* |
| *Resultado da reescrita:* | *RHS* |
| *Posição da estrutura onde a regra é aplicada:* | *P* |
| *Profundidade a que a estrutura se encontra:* | *Lv* |
| *Estratégia aplicada:* | *St = apply_relation_log* |
| *Regra aplicada:* | *R = convert_root* |
| *Contexto ou padrão da regra aplicada:* | *C = root(k,f)* |

```
1  apply_relation(log(Arg1,root(Arg21,Arg22)),RHS,P,Lv,St,R,C):-
2    (
3      any_base(Arg1)
4    ),
5    (
6      degree_k(Arg21)
7    ),
8    (
9      except_zero_and_all_ones(Arg22)
10   ),
11   (
12     positive(Arg22)
13   ),
14   convert(root(Arg21,Arg22),NArg2,_,_,_,R,C),
15   log(Arg1,NArg2,RHS).
```

A regra *β-log_root* , representada aqui, unifica as expressões $\log(a, \text{root}(k, f))$ e $\log(Arg_1, \text{root}(Arg_{21}, Arg_{22}))$. O unificador mais geral (Secção 3.3.2) é a substituição $\sigma = \{Arg_1 \mapsto a, Arg_{21} \mapsto k, Arg_{22} \mapsto f\}$. O objectivo *β-root* está devidamente contextualizado (Programa A.1-5-14) pela regra *β-log_root* .

A resolvente (Secção 3.3.2) da reescrita $\beta$-*root* é simplesmente a substituição $\sigma_1 = \{NArg_2 \mapsto \text{pwr}(\text{div}(1, k_1), k_2)\}$. A resolvente da reescrita $\beta$-*log_root*, $\lambda_\beta(R)\sigma_1$, é simplesmente o resultado da aplicação $([Arg_1, NArg_2]\lambda_\alpha(R))\sigma_1$ ao lado direito, $\lambda_\beta(R)$ da fórmula (1). As variáveis $Arg_1$ e $Arg_{2j}$ são as variáveis livres na transformação condicional $C \mid L \to \lambda_\alpha(R)$, ou seja, as variáveis livres na fórmula

$$\log_{Arg_1} \sqrt[Arg_{21}]{Arg_{22}} \to \log_{Arg_1} NArg_2 \qquad (1)$$

O Programa A.1-6 mostra a implementação Prolog de uma regra de reescrita $\psi$-*der_sum*. As regras para o operador *der* estão mantidas no ficheiro *apply_relation_der.pl*.

**Programa A.1-6** ***Regra de reescrita*** $\psi$*-der_sum*

*Regra:* *LHS -> RHS*
*Lado esquerdo da regra:* *der(Arg1,sum(Arg21,Arg22))*
*Resultado da reescrita:* *RHS*
*Posição da estrutura onde a regra é aplicada:* *P*
*Profundidade a que a estrutura se encontra:* *Lv*
*Estratégia aplicada:* *St = apply_relation_der*
*Regra aplicada:* *R = relate_der_sum*
*Contexto ou padrão da regra aplicada:* *C = der(1,sum(f,g))*

```
1  apply_relation(der(Arg1,sum(Arg21,Arg22)),RHS,P,Lv,St,R,C) :-
2     (
3       R = relate_der_sum,
4       C = der(1,sum(f,g))
5     ),
6     (
7       first_order(Arg1)
8     ),
9     (
```

```
10      variable_or_function(Arg21)
11    ),
12    (
13      except_zero(Arg22)
14    ),
15    sum(NArg1,NArg2,RHS),        % df/dx+dg/dx
16    der(1,Arg21,NArg1),          % df/dx
17    der(1,Arg22,NArg2).          % dg/dx
```

A regra $\psi$-*der_sum*, representada aqui, unifica as expressões $\mathrm{der}(1,\mathrm{sum}(f,g))$ e $\mathrm{der}(Arg_1,\mathrm{sum}(Arg_{21},Arg_{22}))$. O unificador mais geral (Secção 3.3.2) é a substituição $\sigma=\{Arg_1\mapsto 1, Arg_{21}\mapsto f, Arg_{22}\mapsto g\}$. As variáveis $Arg_{ij}$ representam as variáveis livres na transformação condicional $C\,|\,L\rightarrow\lambda_\psi(R)$, ou seja, as variáveis livres na fórmula

$$\frac{\mathrm{d}}{\mathrm{dx}}(Arg_{21}+Arg_{22})\rightarrow\frac{\mathrm{d}}{\mathrm{dx}}Arg_{21}+\frac{\mathrm{d}}{\mathrm{dx}}Arg_{22} \qquad (1)$$

A resolvente (Secção 3.3.2) da reescrita $\psi$-*der_sum*, $\lambda_\psi(R)\sigma$, é simplesmente o resultado da aplicação $([Arg_{21},Arg_{22}]\lambda_\psi(R))\sigma$ ao lado direito $\lambda_\psi(R)$ da fórmula (1).

## A.1.3 Reescrita por Simplificação

A reescrita por simplificação ($\downarrow$-*op*) é executada pelo predicado *simplify* / 12.

O Programa A.1-7 mostra a implementação Prolog de uma regra de reescrita $\downarrow$-*sum*, por recursividade. As regras para o operador *sum* estão mantidas no ficheiro *simplify_sum.pl*.

**Programa A.1-7** ***Regra de reescrita*** $\downarrow$***-sum, por recursividade***

```
  Regra:                          LHS -> RHS
  Lado esquerdo da regra:         sum(Arg1,Arg2)
  Resultado da reescrita:         RHS
  Expressão a resolver:           S
  Profundidade a que a
        reescrita a tem lugar:    Lv
  Posição da estrutura onde a
        reescrita tem lugar:      P
Posição inicial:                  P1
  Estratégia aplicada à
        profundidade Lv:          St
Estratégia aplicada à
        profundidade Lv1:         St1 = simplify_sum_2
  Profundidade inicial:           Lv1
  Regra aplicada à posição P:     R = evaluate_sum
  Contexto ou padrão da
        regra aplicada:           C   = sum(k(1),k(2))
  Flag (tutor/user):              T
```

```
 1  simplify(sum(Arg1,Arg2),RHS,S,Lv,P,P1,St,St1,Lv1,R,C,T):-
 2    (
 3      St=simplify_sum_2
 4    ),
 5    (
 6       compound(Arg2)
 7    ),
 8    (
 9       Lv2 is Lv+1, pos(P,_,P2)
10    ),
11    simplify(Arg2,NArg2,S,Lv2,P2,P1,St2,St1,Lv1,R,C,T),
12    sum(Arg1,NArg2,RHS),
13    except_in(S,sum(Arg1,Arg2),RHS,Lv,P1,St,R,C,T).
```

A regra $\downarrow$-*sum*, representada aqui, unifica as expressões $\text{sum}(f, \text{sum}(k_1, k_2))$ e $\text{sum}(Arg_1, Arg_2)$. O unificador mais geral (Secção 3.3.2) é simplesmente a substituição $\sigma_1 = \{Arg_1 \mapsto f, Arg_2 \mapsto \text{sum}(k_1, k_2)\}$. As variáveis $Arg_i$ são as variáveis livres na transformação condicional $C \mid L \to \lambda_{\downarrow}(R)$, ou seja, as variáveis livres na fórmula

$$Arg_1 + Arg_2 \rightarrow Arg_1 + NArg_2 \quad (1)$$

Como $Arg_2$ é uma expressão composta (Programa A.1-7-6), a estratégia que é sugerida, é uma estratégia de simplificação à direita, ou seja, a aplicação de uma regra de reescrita $\downarrow$-*sum*, por recursividade à direita (Programa A.1-7-11). A variável $Lv_1$[62], define a profundidade (nível) (Secção 4.4.1) a que essa expressão se encontra na estrutura.

O literal seleccionado (Programa A.1-7-11)[63] é uma instância de uma regra de reescrita-$\downarrow$, para a qual não se conhece ainda a regra (ou axioma) a aplicar. A sua resolvente (Secção 3.3.2) é a expressão $NArg_2$. A regra que é eventualmente aplicada é uma regra de reescrita-$\kappa$. Isto porque, a expressão representada por $Arg_2$ é, de facto, um *redex*. A estratégia aplicada é, portanto, uma estratégia de simplificação com redução por cálculo. Podemos dizer que a expressão é simplificável por cálculo. Como vimos na Secção 4.3, que uma estratégia de resolução é identificada pelo tipo de regra que é aplicada (redução, transformação ou simplificação) e pelo nível (profundidade) da estrutura em que é aplicada.

A resolvente (Secção 3.3.2) da reescrita $\downarrow$-*sum*, $\lambda_{\downarrow}(R)\sigma$, é simplesmente o resultado da aplicação $([Arg_1, NArg_2]\lambda_{\downarrow}(R_L))\sigma$ ao lado direito $\lambda_{\downarrow}(R)$ da fórmula (1). A resolvente é, de facto, o resultado de duas substituições, $\sigma_1$ e $\{NArg_2 \mapsto k\}$, ou seja, o resultado de uma composição lógica de dois construtores $\lambda$-*sum*, efectuados a níveis diferentes da estrutura. $k$ é simplesmente o resultado de uma redução, por cálculo, da expressão $\text{sum}(k_1, k_2)$. O pós-requisito (Programa A.1-7-13) exige apenas que o resultado da reescrita não faça parte da memória colectiva

[62] Note que $Lv_1$ representa apenas "*o nível seguinte*". O nível exacto onde essa substituição irá ter lugar depende do contexto onde a expressão está inserida.

[63] Neste caso, foi seleccionada uma regra-$\kappa$ para a classe sum.

da resolução da expressão *S* .

O Programa A.1-8 mostra a implementação Prolog de uma regra de reescrita ↓*-root* , por factorização. As regras para o operador *root* estão mantidas no ficheiro *simplify_root.pl* .

**Programa A.1-8** ***Regra de reescrita*** ↓*-root* ***por factorização***

| | |
|---|---|
| *Regra:* | *LHS -> RHS* |
| *Lado esquerdo da regra:* | *root(Arg1,Arg2)* |
| *Resultado da reescrita:* | *RHS* |
| *Expressão a resolver:* | *S* |
| *Profundidade a que a reescrita a tem lugar:* | *Lv* |
| *Posição da estrutura onde a reescrita tem lugar:* | *P* |
| *Posição inicial:* | *P1* |
| *Estratégia aplicada à profundidade Lv:* | *St* |
| *Estratégia aplicada à profundidade Lv1:* | *St1 = factor_out_root* |
| *Profundidade inicial:* | *Lv1* |
| *Regra aplicada à posição P:* | *R = factor_out_root* |
| *Contexto ou padrão da regra aplicada:* | *C = root(k,k(p))* |
| *Flag (tutor/user):* | *T* |

```
1   simplify(root(Arg1,Arg2),RHS,S,Lv,P,P,St,St,Lv,R,C,T):-
2     (
3       St=factor_out_root
4     ),
5     (
6       both_regular_numbers(Arg1,Arg2)
7     ),
8     (
9       except_zero_and_all_ones(Arg2)
10    ),
11    (
12      Lv1 is Lv+1
13    ),
```

```
14      factorize(root(Arg1,Arg2),NArg2,P,Lv,St,R,C),
15      root(Arg1,NArg2,RHS),
16      except_in(S,root(Arg1,Arg2),RHS,Lv,P,St,R,C,T).
```

A regra $\downarrow$-*root*, representada aqui, unifica as expressões root$(k, k_p)$ e root$(Arg_1, Arg_2)$. O unificador mais geral (Secção 3.3.2) é a substituição $\sigma = \{Arg_1 \mapsto k, Arg_2 \mapsto k_p\}$. A regra de reescrita $\gamma$-*root* (Programa A.1-9), que é executada aqui (Programa A.1-8-14), está devidamente contextualizada pela assinatura *root*.

A resolvente (Secção 3.3.2) da reescrita $\gamma$-*root*, $NArg2$, é simplesmente a substituição $\sigma_1 = \{NArg_2 \mapsto \text{pwr}(k, k_p)\}$. A resolvente da reescrita $\downarrow$-*root*, $\lambda_\gamma(R)\sigma_1$, é simplesmente o resultado da aplicação $([Arg_1, NArg_2]\lambda_\gamma(R))\sigma_1$ ao lado direito $\lambda_\gamma(R)$ da fórmula (1). As variáveis $Arg_i$ são as variáveis livres na transformação condicional, $C \mid L \rightarrow \lambda(R)$, ou seja, as variáveis livres na fórmula

$$\sqrt[Arg_1]{Arg_2} \rightarrow \sqrt[Arg_1]{NArg_2} \qquad (1)$$

## A.1.4 Reescrita Contextualizada

A reescrita contextualizada ($\beta$-*op* e $\gamma$-*op*) é executada pelos predicados *convert*/7 e *factorize*/7.

O Programa A.1-9 mostra a implementação Prolog de uma regra de reescrita contextualizada $\beta$-*root*. As regras $\beta$-*op* são mantido no ficheiro *convert.pl*.

**Programa A.1-9** ***Regra de reescrita contextualizada*** $\beta$-*root*

```
Regra:                          LHS -> RHS
Lado esquerdo da regra:         root(Arg1,Arg2)
Resultado da reescrita:         RHS
Posição da estrutura onde
```

```
        a regra é aplicada:          P
Profundidade a que a
        estrutura se encontra:       Lv
Estratégia aplicada:                 St [64]
Regra aplicada:                      R = convert_root
Contexto ou padrão da
        regra aplicada:              C = root(k,f)

1   convert(root(Arg1,Arg2),RHS,P,Lv,St,R,C) :-
2     (
3       R = convert_root,
4       C = root(k,f)
5     ),
6     (
7       degree_k(Arg1)
8     ),
9     pwr(NArg1,Arg2,RHS), div(1,Arg1,NArg1).
```

A regra $\beta$-*root*, representada aqui, unifica a expressão root$(k, f)$ e root$(Arg_1, Arg_2)$. O unificador mais geral (Secção 3.3.2) é a substituição $\sigma = \{Arg_1 \mapsto k, Arg_2 \mapsto f\}$. As variáveis $Arg_i$ são as variáveis livres na transformação condicional $C \mid L \to \lambda_\beta(R)$, ou seja, as variáveis livres na fórmula

$$\sqrt[Arg_1]{Arg_2} \to \left(Arg_2\right)^{\frac{1}{Arg_1}} \tag{1}$$

A resolvente (Secção 3.3.2) da reescrita $\beta$-*root*, $\lambda_\beta(R)\sigma$, é simplesmente o resultado da aplicação $([Arg_1, Arg_2]\lambda_\beta(R_L))\sigma$ ao lado direito $\lambda_\beta(R)$ da fórmula (1).

O Programa A.1-10 mostra a implementação Prolog de uma regra de reescrita contextualizada $\gamma$-*root*. As regras $\gamma$-*op* são mantido no ficheiro *factorize.pl*.

[64] A estratégia aplicada depende do contexto.

**Programa A.1-10** ***Regra de reescrita contextualizada*** *γ-root*

| | |
|---|---|
| *Regra:* | *LHS -> RHS* |
| *Lado esquerdo da regra:* | *root(Arg1,Arg2)* |
| *Resultado da reescrita:* | *RHS* |
| *Posição da estrutura onde a regra é aplicada:* | *P* |
| *Profundidade a que a estrutura se encontra:* | *Lv* |
| *Estratégia aplicada:* | *St = factor_out_root* |
| *Regra aplicada:* | *R = factor_out_root* |
| *Contexto ou padrão da regra aplicada:* | *C = root(k,k(p))* |

```
1   factorize(root(Arg1,Arg2),RHS,P,Lv,St,R,C):-
2     (
3       R = factor_out_root,
4       C = root(k,k(p))
5     ),
6     (
7       degree_k(Arg1)
8     ),
9     (
10      positive_number(Arg2)
11    ),
12    (
13      except_zero_and_one(Arg2)
14    ),
15    (
16      factored_as(2,Arg2,pwr(ND,D))
17    ),
18    (
19      ND>Arg1;
20      reducible(div(ND,Arg1))
21    ),
22    pwr(ND,D,RHS).
```

A regra *γ-root*, representada aqui, unifica as expressões root($k, k_p$) e root($Arg_1, Arg_2$). O unificador mais geral (Secção 3.3.2) é a substituição $\sigma = \{Arg_1 \mapsto k, Arg_2 \mapsto k_p\}$. As variáveis $Arg_i$ são as variáveis livres na

transformação condicional $C \mid L \rightarrow \lambda_{\gamma}(R)$, ou seja, as variáveis livres na fórmula

$$\sqrt[Arg_1]{Arg_2} \rightarrow \sqrt[Arg_1]{D^{ND}} \quad (1)$$

A resolvente (Secção 3.3.2) da reescrita $\gamma$-*root*, $\lambda_{\gamma}(R)\sigma$, é simplesmente o resultado da aplicação $([Arg_1, Arg_2]\lambda_{\gamma}(R_L))\sigma$ ao lado direito $\lambda_{\gamma}(R)$ da fórmula (1).

## A.1.5 Reescrita por Composição Lógica

A reescrita por composição lógica é executada pelos predicados $'op'/3$.

O Programa A.1-11 mostra a implementação Prolog de uma regra de reescrita contextualizada $\lambda$-*log*. As regras $\lambda$-*op* são mantidas no ficheiro *operators.pl*.

**Programa A.1-11** ***Regra de reescrita contextualizada*** $\lambda$-*log*

| | |
|---|---|
| *Regra:* | *LHS -> RHS* |
| *Lado esquerdo da regra:* | *[Arg1,Arg2]Op* |
| *Resultado da reescrita:* | *log(Arg1,Arg2)* |
| *Construtor:* | *Op* |
| *Argumentos:* | *Arg1 e Arg2* |

```
1   log(Arg1,Arg2,log(Arg1,Arg2)).
```

A regra $\lambda$-*log*, representada aqui, unifica as expressões $\log(S_1, S_2, R)$ e $\log(Arg_1, Arg_2, \log(Arg_1, Arg_2))$. O unificador mais geral (Secção 3.3.2) é a substituição $\sigma_1 = \{Arg_1 \mapsto S_1, Arg_2 \mapsto S_2, R \mapsto \log(S_1, S_2)\}$.

A resolvente (Secção 3.3.2) da reescrita $\lambda$-*log*, $\lambda(R)\sigma_1$, é simplesmente o resultado da aplicação $([Arg_1, Arg_2]\lambda(R))\sigma_1$ ao lado direito $\lambda(R)$ da fórmula (1).

As variáveis $Arg_i$ são as variáveis livres na transformação $L \rightarrow \lambda(R)$, ou seja, as variáveis livres na fórmula

$$[Arg_1, Arg_2]_{\log} \rightarrow \log_{Arg_1} Arg_2 \quad (1)$$

## A.1.6 Reescrita por Comutação

A reescrita por comutação é executada pelo predicado *commute* / 7 .

O Programa A.1-12 mostra a implementação Prolog de uma regra de reescrita contextualizada $\chi$-*sum*. As regras $\chi$-*op* são mantidas no ficheiro *commute.pl* .

**Programa A.1-12** ***Regra de reescrita contextualizada*** $\chi$-*sum*

| | |
|---|---|
| *Regra:* | *LHS -> RHS* |
| *Lado esquerdo da regra:* | *sum(Arg1,Arg2)* |
| *Resultado da reescrita:* | *RHS* |
| *Posição da estrutura onde a regra é aplicada:* | *P* |
| *Profundidade a que a estrutura se encontra:* | *Lv* |
| *Estratégia aplicada:* | *St = apply_relation_sum* |
| *Regra aplicada:* | *R = commute_sum* |
| *Contexto ou padrão da regra aplicada:* | *C = sum(f,g)* |

```
1  commute(sum(Arg1,Arg2),RHS,P,Lv,St,R,C):-
2    (
3      R = commute_sum,
4      C = sum(f,g)
5    ),
6    sum(Arg2,Arg1,RHS).
```

A regra $\chi$-*sum*, representada aqui, unifica as expressões sum($f, g$) e sum($Arg_1, Arg_2$). O unificador mais geral (Secção 3.3.2) é a substituição $\sigma_1 = \{Arg_1 \mapsto f, Arg_2 \mapsto g\}$. A resolvente da reescrita $\chi$-*sum*, $\lambda_\chi(R)\sigma$, é

simplesmente o resultado da aplicação $([Arg_2, Arg_1]\lambda_\chi(R))\sigma$ ao lado direito $\lambda_\chi(R)$ da fórmula (1). As variáveis $Arg_i$ são as variáveis livres na transformação $C\left|L \to \lambda_\chi(R)\right.$, ou seja, as variáveis livres na fórmula

$$Arg_1 + Arg_2 \to Arg_2 + Arg_1 \qquad (1)$$

O Programa A.1-13 mostra a implementação Prolog de uma regra de reescrita $\chi$-*sum*, contextualizada. As regras $\chi$-*sum*, contextualizadas, são mantidas no ficheiro *apply_relation_sum.pl* .

**Programa A.1-13** ***Regra de rescrita*** $\chi$-*sum*, ***contextualizada***

```
Regra:                          LHS -> RHS
Lado esquerdo da regra:         sum(Arg1,Arg2)
Resultado da reescrita:         RHS
Posição da estrutura onde
        a regra é aplicada:     P
Profundidade a que a
        estrutura se encontra:  Lv
Estratégia aplicada:            St = apply_relation_sum
Regra aplicada:                 R  = commute_sum
Contexto ou padrão da
        regra aplicada:         C = sum(f,g)
```

```
 1  apply_relation(sum(Arg1,Arg2),RHS,P,Lv,St,R,C):-
 2    (
 3      % ----- (num,sym)
 4      ( number(Arg1), symbolic_number(Arg2) );
 5      % ----- (const,var)
 6      ( constant(Arg1), variable(Arg2) );
 7      % ----- (const,func)
 8      ( constant(Arg1), function(Arg2) );
 9      % ----- (func,var)
10      ( function(Arg1), variable(Arg2) )
11    ),
12    commute(sum(Arg1,Arg2),RHS,P,Lv,St,R,C).
```

A regra de reescrita $\chi$-*sum*, representada aqui, unifica as expressões, sum($f$, $g$) e sum($Arg_1$, $Arg_2$). O unificador mais geral (Secção 3.3.2) é a substituição $\sigma = \{Arg_1 \mapsto f, Arg_2 \mapsto g\}$.

A resolvente (Secção 3.3.2) da transformação, $\lambda_\chi(R)\sigma$ é, simplesmente, o resultado da aplicação, $([Arg_1, Arg_2]\lambda_\chi(R))\sigma$ ao lado direito $\lambda_\chi(R)$ da fórmula. As variáveis $Arg_i$ são as variáveis livres na transformação condicional $C|L \to \lambda_\chi(R)$, ou seja,

$$Arg_1 + Arg_2 \to Arg_2 + Arg_1$$

A regra de reescrita, $\chi$-*sum* (Programa A.1-12), está aqui devidamente contextualizada (Programa A.1-13-12). Para que ela seja aplicada, é necessário que os termos da soma satisfaçam aos requisitos do programa (Programa A.1-13-2/11), ou seja, que eles sejam expressões dos tipos indicados.

## A.2 Resolução Simbólica de Expressões

Esta secção lista o código que controla a resolução simbólica de expressões matemáticas. A resolução simbólica de expressões matemáticas envolve a análise e execução de frases matemáticas. Todas elas envolvem a simplificação de expressões por reescrita axiomática.

### A.2.1 Processamento de Frases

O processamento de frases matemáticas é baseada nos predicados *query*/1, *parse*/5, *execute*/4 e *process*/5. A análise sintáctica de uma frase matemática é feita utilizando uma gramática DCG e um dicionário (léxico) Prolog. A gramática e o dicionário são mantidas no ficheiro *parse.pl*.

O Programa A.2-1 mostra a implementação Prolog do predicado *query* / 1 . O predicado é mantido no ficheiro *parse.pl* .

**Programa A.2-1** ***Processamento de uma frase matemática***

*Frase matemática:* *Ws*

```
1   query(Ws):-
2     parse(Ws,PN,S,NS,Err),
3     query_result(PN,S,E,R,LS,NS,St,RA,C,Err),
4     write_query(PN,S,E,R,LS,NS,St,RA,C,Err).
```

O predicado *parse* / 5 analisa e executa a frase.

O Programa A.2-2 mostra a implementação Prolog do predicado *parse* / 5 . O predicado é mantido no ficheiro *parse.pl* .

**Programa A.2-2** ***Análise e execução de uma frase matemática***

*Frase matemática:* *Ws*
*Número atribuído à sessão prática:* *P*
*Expressão a resolver:* *S*
*Número de passos resolvidos:* *L*
*Mensagem enviada:* *Err*

```
1   parse(Ws,N,S,L,Err):-
2     ( s(S,Ws,[]) ->
3       process(N,Ws,S,L,Err);
4       get_practice(N),
5       no_steps(L),
6       S = none,
7       Err = invalid_sentence
8     ).
```

O predicado *s* / 3 analisa a frase, sintacticamente, e converte a frase numa expressão lógica. O predicado *process* / 5 executa essa expressão.

O Programa A.2-3 mostra a implementação Prolog do predicado

*process* / 5 . O predicado é mantido no ficheiro *parse.pl* .

**Programa A.2-3** ***Execução de uma frase matemática***

```
Sessão prática:          N
Frase matemática:        Ws
Expressão a resolver:    resolve(practice(expression(Arg)))
Último passo resolvido:  L
Mensagem enviada:        Err

 1  process(N,Ws,resolve(practice(expression(Arg))),L,Err):-
 2    set_next_practice(N),
 3    make_practice(P,N),
 4    ( practice(P,L) ->
 6      Err = practice_in_use;
 7      ( expression_in(P,Arg) ->
 8        last_step(P,_,Last),
 9        ( Last = 0 ->
10          step(P,0),
11          ( practice(P,L) ->
12            Err = practice_resolved,
13            execute(P,L,L,view(P,L));
14            Err = practice_to_be_resolved,
15            last_step(P,_,L),
16            execute(P,L,L,view(P,L))
17          );
18          Err = practice_to_be_resolved,
19          last_step(P,_,L),
20          execute(P,1,L,view(P,1))
21        );
22        assert_expression(P,Arg),
23        assert_problem(Arg,Ws),
24        ( expression_in(P,_) ->
25          step(P,0),
26          ( practice(P,L) ->
27            Err = practice_resolved,
28            execute(P,L,L,view(P,L));
29            Err = practice_to_be_resolved,
30            last_step(P,_,L),
31            execute(P,L,L,view(P,L))
32          );
33          Err = missing_expression,
```

```
34              no_steps(L)
35          )
36       )
37    ).
```

Se a prática *P* não está ainda resolvida ( *practice* / 2 ) (Programa A.2-3-4) mas a expressão está definida (Programa A.2-3-7), então o primeiro passo é resolvido ( *step* / 2 ) (Programa A.2-3-10) se estiver ainda por resolver (Programa A.2-3-9). Caso contrário, a expressão é definida primeiro ( *assert_expression* / 2 ) (Programa A.2-3-22). A prática é então visualizada (Programa A.2-3-13)

Se o primeiro passo já está por resolvido (Programa A.2-3-9), a prática é simplesmente visualizada (Programa A.2-3-16, 20, 28 e 31).

## A.2.2 Simplificação de Expressões

A simplificação de expressões matemáticas é baseada nos predicados *resolve* / 2 , *step* / 2 e *simplify* / 12 .

O Programa A.2-4 mostra a implementação Prolog do método de simplificação de expressões. O método é mantido no ficheiro *resolve.pl* .

**Programa A.2-4** ***A resolução simbólica de expressões***

```
Nome da prática:                P
Expressão a resolver:           S
Número do passo:                N
Lado esquerdo do passo:         LHS
Profundidade da estrutura:      Lv
Posição da estrutura:           Pos

resolve(P,N):-
   ( practice(P,_) ->
      notify(P,N,step_resolved);
      ( expression_in(P,S) ->
         ( step(P,S,N,LHS,RHS,0,Pos,Lv,St,R,C) ->
            Next is N+1,
```

```
7           ( step(P,S,Next,_,_,0,_,_,_,_,_) ->
8             notify(P,N,step_resolved);
9             simplify(P,S,N,RHS)
10          );
11          notify(P,N,invalid_step)
12        );
13        notify(P,N,missing_expression)
14    )
15  ).
16  simplify(P,S,N,LHS):-
17    max_steps(Max_steps),
18    ( simplify(LHS,RHS1,S,0,0,Pos,St2,St,Lv,R,C,user) ->
19      ( Next is N+1,
20        assert_step(P,S,Next,LHS,RHS1,0,Pos,Lv,St,R,C),
21        resolve(P,Next)
22      );
23      ( included(S,_,_,_,LHS,_,RHS2,_,_,_) ->
24        ( LHS \= RHS2 ->
25          complete(P,S,N,RHS2);
26          terminate(P)
27        );
28        ( common(S,_,_,_,LHS,_,RHS2,_,_,_) ->
29          ( LHS \= RHS2 ->
30            complete(P,S,N,RHS2);
31            terminate(P)
32          );
33          terminate(P)
34        )
35      )
36    ).
```

O predicado *resolve* / 2 (Programa A.2-4-1) resolve simbolicamente a expressão *RHS* , obtida no passo $N-1$, aplicando-lhe uma estratégia *simplify* / 12 (Programa A.2-5). O predicado *simplify* / 4 (Programa A.2-4-16) aplica a estratégia. Se a reescrita é efectuada com sucesso (Programa A.2-4-19 a 22), o passo $N$, $LHS \rightarrow RHS$, é adicionado à memória colectiva da resolução da expressão $S$ (Programa A.2-4-20), e o resultado, *RHS1*, resolvido como o passo seguinte, $N+1$, dessa resolução (Programa A.2-4-21). Caso contrário, a solução é

completada a partir da memória (Programa A.2-4-25 ou 30).

O Programa A.2-5 mostra a implementação Prolog de uma estratégia de resolução de expressões. As estratégias de resolução são baseadas no predicado *simplify* / 12 . As estratégias de resolução para o operador *root* estão mantidas no ficheiro *simplify_root.pl* .

**Programa A.2-5** ***A aplicação de uma estratégia de resolução***

```
Regra:                          LHS -> RHS
Lado esquerdo da regra:         root(Arg1,Arg2)
Resultado da reescrita:         RHS
Expressão a resolver:           S
Profundidade a que a
        reescrita a tem lugar:  Lv
Posição da estrutura onde a
        reescrita tem lugar:    P
Posição inicial:                P1
Estratégia aplicada à
        profundidade Lv:        St
Estratégia aplicada à
        profundidade Lv1:       St1 = apply_relation_root
Profundidade inicial:           Lv1
Regra aplicada à posição P:     R [65]
Contexto ou padrão da
        regra aplicada:         C
Flag (tutor/user):              T
```

```
1   simplify(root(Arg1,Arg2),RHS,S,Lv,P,P,St,St,Lv,R,C,T):-
2     (
3       St=apply_relation_root
4     ),
5     apply_relation(root(Arg1,Arg2),RHS,P,Lv,St,R,C),
6     except_in(S,root(Arg1,Arg2),RHS,Lv,P,St,R,C,T).
```

O predicado *simplify* / 12 (Programa A.2-5-1) aplica uma estratégia de simplificação por conversão axiomática, ↓-*apply_relation_root* , ou seja, uma

[65] A regra aplicada depende da regra que for escolhida e executada com sucesso.

regra de reescrita da classe *root*, baseada no predicado *apply_relation*/7 (Programa A.2-5-5). Como sabemos, as regras de reescrita estão organizadas por ordem decrescente do seu poder redutor. Por exemplo, uma regra do tipo $\sqrt[k]{1} \to 1$ possui um poder redutor mais elevado que uma regra do tipo $\sqrt[k]{fg} \to \sqrt[k]{f}\sqrt[k]{g}$.

## A.3 Requisitos Operacionais

Esta secção lista o código que é utilizado na determinação da afinidade operacional e o princípio de exclusão.

### A.3.1 Afinidade Operacional

A afinidade operacional entre operadores e argumentos é determinada pelos predicados *akin_number*/3, *akin_variable*/3 e *akin_function*/3. O predicado *akin*/3 é apenas uma redefinição generalizada desses 3 predicados.

O Programa A.3-1 mostra a implementação Prolog do predicado *akin*/3. O predicado é mantido no ficheiro *akin.pl*.

**Programa A.3-1** ***Afinidade operacional***

```
Expressão:          Op(Arg1,Arg2)
Contexto:           Op
1º Argumento:       Arg1
2º Argumento:       Arg2

1   akin(Op,Arg1,Arg2):-
2     akin_number(Op,Arg1,Arg2);
3     akin_variable(Op,Arg1,Arg2);
4     akin_function(Op,Arg1,Arg2).
```

*Op* é o operador matemático cujos argumentos $Arg_i$ se pretende analisar do ponto de vista operacional. O operador define o contexto em que os seus argumentos são analisados.

O Programa A.3-2 mostra a implementação Prolog de uma regra *op_sum_prod* , baseada no predicado *akin_function* / 3 , que determina a afinidade operacional entre somas e diferenças, no contexto de somas e diferenças. Essas regras são mantidas no ficheiro *akin.pl* .

**Programa A.3-2** ***Afinidade operacional entre somas e produtos***

```
Expressão:          sum(Arg1,Arg2) ou diff(Arg1,Arg2)
Contexto:           sum ou diff
1º Argumento:       sum(Arg11,Arg12)
2º Argumento:       prod(Arg21,Arg22)
```

```
1  akin_function(Op,sum(Arg11,Arg12),prod(Arg21,Arg22)):-
2    (
3      Op=sum; Op=diff
4    ),
5    except_both_constants(sum(Arg11,Arg12),prod(Arg21,Arg22)),
6    except_zero(Arg12),
7    except_zero_and_all_ones(Arg21),
8    akin_variable_or_function(Op,Arg12,Arg22).
```

## A.3.2 Memorização

A memorização de uma reescrita é baseada no predicado *assert_step* / 11 . A reescrita é registada como uma instância do predicado *step* / 11 . A memória colectiva de uma resolução é utilizada pelo princípio de exclusão (ver o predicado *except_in* / 9 ).

O Programa A.3-3 mostra a implementação Prolog de como é feita a memorização de uma reescrita. A regra é baseada no predicado *assert_step* / 11 . O predicado é mantido no ficheiro *assert.pl* .

**Programa A.3-3** ***A memorização de uma reescrita***

| | |
|---|---|
| *Nome da prática:* | *P* |
| *Expressão a resolver:* | *S* |
| *Número do passo resolvido:* | *N* |
| *Lado esquerdo do passo:* | *LHS* |
| *Lado direito do passo:* | *RHS* |
| *Topo da estrutura:* | *Lv* |
| *Posição da estrutura:* | *Pos* |
| *Profundidade da estrutura:* | *D* |
| *Estratégia aplicada:* | *St* |
| *Regra aplicada:* | *R* |
| *Contexto ou padrão da regra aplicada:* | *C* |

```
1   assert_step(P,S,N,LHS,RHS,Lv,Pos,D,St,R,C):-
2     ( step(P,S,N,LHS,RHS,Lv,Pos,D,St,R,C) ->
3       continue;
4       assert(step(P,S,N,LHS,RHS,Lv,Pos,D,St,R,C)),
5       ( last_step(P,_,_) ->
6         retract(last_step(P,_,_)),
7         assert(last_step(P,N,N));
8         assert(last_step(P,N,N))
9       )
10    ).
```

O resultado de uma reescrita axiomática, $LHS \rightarrow RHS$, é memorizado como uma instância do predicado *step*/11 (Programa A.3-3-4).

O Programa A.3-4 mostra a implementação Prolog de como é consultada a memória colectiva de uma resolução. O princípio de exclusão é exercido com base nessa consulta (ver o predicado *simplify*/12). A consulta da memória colectiva é baseada no predicado *except_in*/9. O predicado é mantido no ficheiro *resolve.pl*.

**Programa A.3-4** ***A consulta da memória colectiva***

| | |
|---|---|
| *Expressão a resolver:* | *S* |
| *Reescrita efectuada:* | *LHS -> RHS* |
| *Resultado anterior:* | *LHS* |
| *Resultado da reescrita:* | *RHS* |
| *Profundidade a que a reescrita* | |

```
        teve lugar:                 Lv
Posição onde a reescrita
        teve lugar:                 P
Estratégia aplicada:                St
Regra aplicada:                     R
Contexto ou padrão da
        regra aplicada:             C
Flag (tutor/user):                  T

 1  except_in(S,LHS,RHS,Lv,P,St,R,C,T):-
 2    ( T = tutor ->
 3      (
 4        assert_common(S,T,LHS,'=>',RHS,Lv,P,St,R,C),
 5        except(solution(S,_,RHS,_,_,_,_,_,_,_)),
 6        except(solution(S,_,LHS,RHS,_,_,_,_,_,_)),
13        ( Lv = 0 ->
14          assert_included(S,T,LHS,'=>',RHS,Lv,P,St,R,C);
15          continue
16        )
17      );
18      (
19        assert_common(S,T,LHS,'=>',RHS,Lv,P,St,R,C),
20        except(step(_,S,_,RHS,_,_,_,_,_,_)),
21        except(step(_,S,_,LHS,RHS,_,_,_,_,_)),
22        ( Lv = 0 ->
23          assert_included(S,T,LHS,'=>',RHS,Lv,P,St,R,C);
24          continue
25        )
26      )
27    ).
```

O predicado *except_in*/9 (Programa A.3-4-1) verifica se a reescrita, $LHS \rightarrow RHS$, já faz parte da memória colectiva da resolução da expressão $S$ (ver predicado *step*/11) e regista as intercepções (Programa A.3-4-4/14 e19/23).

O princípio de exclusão é exercido com base na consulta dessa memória. Se a reescrita já faz parte da memória colectiva da resolução (Programa A.3-4-4/5 e 20/21), a reescrita é rejeitada (ver o predicado *simplify*/12) e uma nova estratégia escolhida, por *backtracking*. Caso contrário, a reescrita é adicionada à memória

colectiva e o passo seguinte resolvido (ver os predicados *assert_step* / 11 e *resolve* / 2 ) .

# Apêndice B A BASE AXIOMÁTICA

## B.1 O Sistema Numérico

Nesta secção estão listadas as propriedades algébricas do sistema numérico que serviram de base à transformação lógica de expressões matemáticas, por via axiomática. O sistema de números reais $\mathcal{R}$, que serviu de sistema numérico para esta tese, é constituído pelo números inteiros $\mathcal{I} = \mathbb{N} \bigcup \mathcal{I}^-$, os números racionais $\mathcal{Q}$ e os números irracionais $\overline{\mathcal{Q}}$. Com excepção dos números naturais $\mathbb{N}$, todos os outros estão representados, nesta tese, por constantes simbólicas. A base axiomática, utilizada nesta tese, é constituída pelos axiomas e teoremas básicos que descrevem as propriedades algébricas do sistema $\mathcal{R}$.

As propriedades algébricas do sistema $\mathcal{R}$ estão listadas neste Apêndice e estão referidas em [AYR65, BEL66, COH03].

### B.1.1 Números Reais

Seja $\mathcal{R}$ o conjunto dos números reais.

**Propriedade B.1.1** *Fecho*

*Para todo o* $x, y \in \mathcal{R}$,

*i)* $x + y \in \mathcal{R}$

*ii)* $x \cdot y \in \mathcal{R}$.

**Propriedade B.1.2** *Comutatividade*

*Para todo o* $x, y \in \mathcal{R}$,

*i)* $x + y = y + x$

*ii)* $x \cdot y = y \cdot x$.

**Propriedade B.1.3** ***Associatividade***

*Para todo o* $x, y, z \in \mathcal{R}$,

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

*ii)* $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.

**Propriedade B.1.4** ***Distributividade***

*Para todo o* $x, y, z \in \mathcal{R}$,

*i)* $x \cdot (y + z) = x \cdot y + x \cdot z$ *(à esquerda)*

*ii)* $(x + y) \cdot z = x \cdot z + y \cdot z$. *(à direita)*

**Propriedade B.1.5** ***Identidade Aditiva***

*Para todo o* $x \in \mathcal{R}$, *existe um único elemento* $0 \in \mathcal{R}$, *tal que* $x + 0 = 0 + x = x$.

A identidade aditiva é também conhecida por elemento neutro da soma.

**Propriedade B.1.6** ***Identidade Multiplicativa***

*Para todo o* $x \in \mathcal{R}$, *existe um único elemento* $1 \in \mathcal{R}$, *tal que* $1 \cdot x = x \cdot 1 = x$.

A identidade multiplicativa é também conhecida por elemento neutro do produto.

**Propriedade B.1.7** ***Inverso Aditivo***

> *Para todo o* $x \in \mathcal{R}$, *existe um único elemento* $-x \in \mathcal{R}$, *tal que* $x + (-x) = (-x) + x = 0$.

O inverso aditivo é também conhecido por elemento simétrico da soma.

**Propriedade B.1.8** ***Inverso Multiplicativo***

> *Para todo o* $x \in \mathcal{R}$, *com* $x \neq 0$, *existe um único elemento* $x^{-1} \in \mathcal{R}$, *tal que* $x \cdot x^{-1} = x^{-1} \cdot x = 1$.

O inverso multiplicativo é também conhecido por elemento simétrico do produto.

As propriedades Propriedade B.1.1 a Propriedade B.1.8 definem os axiomas do corpo $\mathcal{R}$ para a soma e o produto.

**Teorema B.1-1**

> *Para todo o* $x, y \in \mathcal{R}$, *tem-se que*
>
> *i)* $x \cdot 0 = 0$;
>
> *ii)* $-x = (-1) \cdot x$;
>
> *iii)* $-0 = 0$;
>
> *iv)* $1^{-1} = 1$;
>
> *v)* $(-1)^{-1} = -1$;
>
> *vi)* $(-x) \cdot (-y) = x \cdot y$;
>
> *vii)* *Se* $x \neq 0$, *então* $(x^{-1})^{-1} = x$;
>
> *viii)* *Se* $x \neq 0$ *e* $y \neq 0$, *então* $(x \cdot y)^{-1} = x^{-1} \cdot y^{-1}$;

A demonstração pode ser encontrada em [COH03].

**Definição B.1-1** ***Diferença***

> *Seja* $x, y \in \mathcal{R}$. *O operador diferença é definido por* $x - y = x + (-y)$, *onde* $-y$ *é o inverso aditivo de* $y$.

**Definição B.1-2** ***Quociente***

> *Seja* $x, y \in \mathcal{R}$. *O operador quociente é definido por* $\frac{x}{y} = xy^{-1}$, *onde* $y^{-1}$ *é o inverso multiplicativo de* $y$.

**Definição B.1-3** ***Potência***

> *Seja* $x \in \mathcal{R}$ *e* $n \in \mathbb{N}$. *O operador potência é definido, recursivamente, por*
>
> *i)* *Se* $x \neq 0$ *e* $n = -1$, *então* $x^{-1}$ *é o inverso multiplicativo de* $x$;
>
> *ii)* *Se* $x \neq 0$ *e* $n \neq -1$, *então*
>
> $$x^n = \begin{cases} x^{n-1} \cdot x & \text{se } n \geq 1, \\ 1 & \text{se } n = 0, \\ (x^{-1})^{-n} & \text{se } n < -1 \end{cases}$$
>
> *iii)* *Se* $x = 0$, *então*
>
> $$0^n = \begin{cases} 0 & \text{se } n \geq 1, \\ \textbf{indeterminado} & \text{se } n \leq 0 \end{cases}$$

Note que $x^{-n} = (x^n)^{-1}$ é a inversa de $x^n$.

**Teorema B.1-2**

> *Seja* $x, y \in \mathcal{R}$, *com* $x \neq 0$ *e* $y \neq 0$, *e* $m, n \in \mathbb{N}$. *Então, tem-se que*
>
> *i)* $(x \cdot y)^m = x^m \cdot y^m$;

*ii)* $x^m \cdot x^n = x^{m+n}$;

*iii)* $(x^m)^n = x^{m \cdot n}$;

A demonstração pode ser encontrada em [COH03].

**Definição B.1-4** ***Raiz***

*Seja* $x \in \mathcal{R}$ *e* $n \in \mathbb{N}$, *com* $x \geq 0$ *e* $n > 0$. *O operador raiz é definido por* $\sqrt[n]{x} = x^{\frac{1}{n}}$ *e tal que* $(\sqrt[n]{x})^n = x$.

**Definição B.1-5** ***Potência de Expoente Racional***

*Seja* $x \in \mathcal{R}$ *e* $m, n \in \mathbb{N}$, *com* $x > 0$ *e* $n > 0$. *Então* $(\sqrt[n]{x})^m = x^{\frac{m}{n}}$

## B.1.2 Números Inteiros

Seja $\mathcal{I}$ o conjunto dos números inteiros. Nesta tese, os números inteiros estão representados por constantes simbólicas.

**Definição B.1-6** ***Valor Absoluto***

*Seja* $p \in \mathcal{I}$. *O valor absoluto de* $p$ *é definido por*

$$|p| = \begin{cases} p & \text{se } p \geq 0, \\ -p & \text{se } p < 0 \end{cases}$$

Note que $\sqrt{p^2} = |p|$.

**Definição B.1-7** ***Divisor ou Factor***

*Seja* $p, q \in \mathcal{I}$ *e* $p \neq 0$. *Diz-se que* $p$ *é o divisor (factor) de* $q$, *e escreve-se* $p \mid q$, *se existe* $r \in \mathcal{I}$ *tal que* $q = p \cdot r$. *Diz-se também que* $q$ *é um múltiplo inteiro de* $p$.

**Definição B.1-8** ***Primo***

> *Seja* $p \in \mathcal{I}$ *e* $p \neq 0, \pm 1$. *Diz-se que* $p$ *é primo se e só se os seus únicos divisores forem* $\pm 1$ *e* $\pm p$.

**Postulado B.1-1**

> *Se* $p \in \mathcal{I}$, *então* $-p$ *é primo se e só se* $p$ *for primo.*

**Definição B.1-9** ***Composto***

> *Seja* $p, q, r \in \mathcal{I}$. *Diz-se que* $p = q \cdot r$ *é composto se* $|q| > 1$ *e* $|r| > 1$.

**Postulado B.1-2**

> *Se '* $p \in \mathcal{I}$ *e* $p \neq 0, \pm 1$, *então* $p$ *é primo ou composto.*

**Definição B.1-10** ***Maior Divisor Comum***

> *Seja* $p, q, r \in \mathcal{I}$. *Diz-se que* $p$ *é o máximo divisor comum de* $q$ *e* $r$ *se* $p \mid q$ *e* $p \mid r$ *e todos os divisores de* $q$ *e* $r$ *forem também divisores de* $p$.

**Teorema B.1-3** ***Factorização em Primos***

> *Todo e qualquer* $p \in \mathcal{I}$, *com* $p > 1$, *possui uma única factorização em primos, que difere apenas na ordem dos seus factores, ou seja,*
>
> $$p = \prod_{i=1}^{n} p_i^{\alpha_i}$$
>
> *onde* $p_i$ *são primos distintos e* $\alpha_i \geq 1$, *o número de factores de cada tipo.*

A demonstração pode ser encontrada em [BEL66]. Note que se $p$ for primo, o produto contem apenas um único factor.

**Definição B.1-11** ***Paridade***

> *Seja* $p,q \in \mathcal{I}$ *com* $p>0$ *e* $q>0$.
>
> *i)* *Se* $p=2q$, $p$ *é par;*
>
> *ii)* *Caso contrário,* $p$ *é ímpar.*

**Teorema B.1-4**

> *Seja* $p,q \in \mathcal{I}$ *com* $p>0$ *e* $q>0$. *Se* $p$ *é ímpar, então* $p=1$ *ou* $p=2q+1$.

A demonstração pode ser encontrada em [BEL66].

**Lema B.1.2-1**

> *Seja* $p \in \mathcal{I}$.
>
> *i)* *Se* $p$ *é par, então* $p^2$ *é par.*
>
> *ii)* *Se* $p$ *é ímpar, então* $p^2$ *é ímpar.*

## B.1.3 Números Racionais

Seja $\mathcal{Q}$ o conjunto dos números racionais. Nesta tese são apenas consideradas as fracções $\frac{m}{n}$, onde $m,n \in \mathcal{I}$ e $n \neq 0$.

**Definição B.1-12** ***Fracção***

> *Seja* $p,q \in \mathbb{N}$ *e* $q \neq 0$. *Uma fracção é um par ordenado* $p/q \in \mathcal{Q}$.

**Postulado B.1-3**

> *Existe uma colecção de objectos designados por racionais positivos.*
> *Cada um destes objectos é representado por uma fracção.*

**Definição B.1-13** ***Igualdade de Fracções***

> *Seja* $m/n, p/q \in \mathcal{Q}$. *Então* $m/n = p/q$ *se e só se* $m \cdot q = p \cdot n$.

**Postulado B.1-4** ***Soma de Fracções***

> *Dado* $m/n, p/q \in \mathcal{Q}$, *existe um número racional positivo,* $m/n + p/q$, *designado por soma, dado por*
>
> $$\frac{m}{n} + \frac{p}{q} = \frac{m \cdot q + p \cdot n}{n \cdot q}$$

**Postulado B.1-5** ***Produto de Fracções***

> *Dado* $m/n, p/q \in \mathcal{Q}$, *existe um número racional positivo,* $(m/n) \cdot (p/q)$, *designado por produto, dado por*
>
> $$\left(\frac{m}{n}\right) \cdot \left(\frac{p}{q}\right) = \frac{m \cdot p}{n \cdot q}$$

**Teorema B.1-5**

> *Para cada* $m/n \in \mathcal{Q}$, *existe um e um só inverso multiplicativo,* $\left(\frac{m}{n}\right)^{-1}$, *dado por* $\left(\frac{m}{n}\right)^{-1} = \frac{n}{m}$.

## B.1.4 Números Irracionais

Seja $\overline{\mathcal{Q}}$ o conjunto dos números irracionais. Nesta tese, os números irracionais são representados por constantes especiais ou simbólicas, como $e$, $\sqrt{2}$ ou $\log_e 2$.

**Postulado B.1-6**

> *Existe uma colecção de objectos designados por reais não*

> *negativos. Cada um destes objectos é representado por uma sequência de intervalos racionais fechados, inclusos.*

**Definição B.1-14** ***Número Irracional***

> *Qualquer número real* $x \in \mathcal{R}$, *não negativo, que não seja racional* $x \notin \mathcal{Q}$, *é irracional, isto é,* $\overline{\mathcal{Q}} = \mathcal{R} \setminus \mathcal{Q}$.

## B.2 Logaritmos e Exponenciais

Nesta secção estão listadas as propriedades dos logaritmos e das exponenciais que serviram de base à reescrita axiomática desses operadores.

### B.2.1 Logaritmos

**Definição B.2-1** ***Número de Néper***

> *Seja* $x \in \mathcal{R}$ *e* $x > 0$. *O número de Néper* $e$ *é o valor* $x$, *único, tal que* $\log x = 1$.

**Definição B.2-2** ***Logaritmo de Base a***

> *Seja* $x, y, a \in \mathcal{R}$, *com* $y > 0$, $a > 0$ *e* $a \neq 1$. *O logaritmo de* $y$ *na base* $a$ *é definido por* $\log_a y$ *tal que* $y = a^x = e^{x \cdot \log_e a}$.

**Teorema B.2-1**

> *O logaritmo natural é o logaritmo na base e, isto é,*
>
> $$\log \equiv \log_e$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.2-2** ***Logaritmo de uma Exponencial***

*Seja* $x, y \in \mathcal{R}$ *e* $x > 0$. *Então, tem-se que*

$$\log(x^y) = y \cdot \log x$$

A demonstração pode ser encontrada em [BEL66]. Note que $\log(e^y) = y \cdot \log e = y$ é um caso particular.

**Teorema B.2-3** ***Soma de Logaritmos***

*Seja* $x, y \in \mathcal{R}$, *com* $x > 0$ *e* $y > 0$. *Então, tem-se que*

$$\log(x \cdot y) = \log x + \log y$$

A demonstração pode ser encontrada em [BEL66].

**Corolário B.2-1**

*Seja* $x \in \mathcal{R}$, *com* $x > 0$. *Então, tem-se que*

$$\log\left(\frac{1}{x}\right) = -\log x$$

A demonstração pode ser encontrada em [BEL66].

**Corolário B.2-2**

*Seja* $x \in \mathcal{R}$ *e* $k \in \mathcal{I}$, *com* $x > 0$. *Então, tem-se que*

$$\log x^k = k \cdot \log x$$

A demonstração pode ser encontrada em [BEL66].

**Corolário B.2-3**

*Seja* $x \in \mathcal{R}$ *e* $k \in \mathcal{I}$, *com* $x > 0$. *Então, tem-se que*

$$\log x^{\frac{1}{k}} = \frac{1}{k} \cdot \log x$$

A demonstração pode ser encontrada em [BEL66].

**Corolário B.2-4**

*Seja* $x \in \mathcal{R}$, *e* $r \in \mathcal{Q}$, *com* $x > 0$. *Então, tem-se que*

$$\log x^r = r \cdot \log x$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.2-4** ***Conversão de Base***

*Seja* $x, a \in \mathcal{R}$, *com* $x > 0$, $a > 0$ *e* $a \neq 1$. *Então, tem-se que*

$$\log_a x = \frac{1}{\log_e a} \cdot \log_e x$$

A demonstração pode ser encontrada em [BEL66].

## B.2.2 Exponenciais

**Definição B.2-3** ***Exponencial Natural***

*Seja* $x \in \mathcal{R}$ *e* $x > 0$. *A exponencial natural de* $x$ *é definida por* $e^x = \text{inv} \log x$, *onde* inv *representa o operador inversão.*

**Definição B.2-4** ***Exponencial***

*Seja* $x, y \in \mathcal{R}$ *e* $x > 0$. *A exponencial de base* $x$, *qualquer, é definida por* $x^y = e^{y \cdot \log x}$.

**Teorema B.2-5**

*Seja* $x, y \in \mathcal{R}$. *Então, tem-se que* $e^{x+y} = e^x \cdot e^y$.

A demonstração pode ser encontrada em [BEL66].

**Teorema B.2-6**

> *Seja* $x, y, z \in \mathcal{R}$ *e* $x > 0$. *Então, tem-se que* $x^{y+z} = x^y \cdot x^z$.

A demonstração pode ser encontrada em [BEL66].

**Corolário B.2-5**

> *Seja* $x, y \in \mathcal{R}$ *e* $x > 0$. *Então, tem-se que* $x^{-y} = \frac{1}{x^y}$.

A demonstração pode ser encontrada em [BEL66].

**Teorema B.2-7**

> *Seja* $x, y, z \in \mathcal{R}$ *e* $x > 0$. *Então, tem-se que* $(x^y)^z = x^{y \cdot z}$.

A demonstração pode ser encontrada em [BEL66].

## B.3 Derivadas

Nesta secção estão listadas as propriedades das derivadas.

**Definição B.3-1** ***Operador Derivada***

> *Designamos o operador derivada,* $\frac{d}{dx}$, *por* $D_x$.

Como notação, optámos por $D_x f$ em vez de $D_x f(x)$ ou $(D_x f)(x)$.

**Teorema B.3-1** ***Linearidade do Operador Derivada***

> *Seja* $f, g : \mathcal{R} \to \mathcal{R}$, *diferenciáveis, com o mesmo domínio* $\mathcal{D}_f = \mathcal{D}_g$, *e* $c \in \mathcal{R}$. *Então* $f + g$ *e* $c \cdot f$ *são diferenciáveis e as suas derivadas dadas por*

$$D_x(f+g) = D_x f + D_x g$$

$$D_x(c \cdot f) = c \cdot D_x f$$

A demonstração pode ser encontrada em [BEL66].

**Definição B.3-2** ***Derivada de Ordem n***

*Seja* $f : \mathcal{R} \to \mathcal{R}$ *diferenciável e* $n \in \mathbb{N}$. *A derivada de ordem* $n$ *de* $f$ *é definida, de forma indutiva, e desde que as operações sejam significativas, por*

*i)* $D_x^{\,0} f = f$

*ii)* $D_x^{\,n} f = D_x\left(D_x^{\,n-1} f\right)$

Note que $D^n = D \circ D^{n-1}$ onde o operador $\circ$ é o operador composição.

**Teorema B.3-2** ***Derivada do Produto***

*Seja* $f, g : \mathcal{R} \to \mathcal{R}$, *diferenciáveis, com o mesmo domínio* $\mathcal{D}_f = \mathcal{D}_g$. *Então* $f \cdot g$ *é diferenciável e a sua derivada dada por*

$$D_x(f \cdot g) = f \cdot D_x g + D_x f \cdot g$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-3** ***Derivada da Inversa***

*Seja* $f : \mathcal{R} \to \mathcal{R}$, *diferenciável, com* $f(x) \neq 0$. *Então* $\frac{1}{f}$ *é diferenciável e a sua derivada dada por*

$$D_x\left(\frac{1}{f}\right) = -\frac{1}{f^2} \cdot D_x f$$

A demonstração pode ser encontrada em [BEL66].

**Corolário B.3-1** ***Derivada do Quociente***

> *Seja* $f, g : \mathcal{R} \to \mathcal{R}$ *, diferenciáveis, com o mesmo domínio* $\mathcal{D}_f = \mathcal{D}_g$ *, e* $g(x) \neq 0$ *. Então* $\frac{f}{g}$ *é diferenciável e a sua derivada dada por*
>
> $$D_x\left(\frac{f}{g}\right) = \frac{D_x f \cdot g - f \cdot D_x g}{g^2}$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-4** ***Derivada da Composta***

> *Seja* $f, g : \mathcal{R} \to \mathcal{R}$ *com o* $\mathcal{CD}_g \subseteq \mathcal{D}_f$ *. Se* $g$ *for diferenciável em* $x$ *e* $f$ *diferenciável em* $g(x)$*, então a composição* $f(g)$ *é diferenciável em* $x$ *com*
>
> $$D_x f(g) = D_g f \cdot D_x g$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-5** ***Derivada da Potência*** $x^n$

> *Seja* $x$ *a função identidade sobre reais,* $id : \mathcal{R} \to \mathcal{R}$ *e* $n \in \mathcal{I}$ *. Se*
>
> *i)* $n > 0$ *, ou*
>
> *ii)* $n < 0$ *e* $x \neq 0$ *,*
>
> *então* $x^n$ *é diferenciável e a sua derivada dada por*
>
> $$D_x x^n = n \cdot x^{n-1}$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-6** ***Derivada da Potência*** $x^{\frac{1}{n}}$

> *Seja* $x$ *a função identidade sobre reais,* $id : \mathcal{R} \to \mathcal{R}$, $n \in \mathcal{I}$ *e* $n \neq 0$. *Se*
>
> *i)* $n$ *é par e* $x > 0$, *ou*
>
> *ii)* $n$ *é ímpar e* $x \neq 0$,
>
> *então* $\sqrt[n]{x} \equiv x^{1/n}$ *é diferenciável e a sua derivada dada por*
>
> $$D_x \sqrt[n]{x} = D_x x^{\frac{1}{n}} = \frac{1}{n} x^{\frac{1}{n}-1} \equiv \frac{1}{n} \frac{\sqrt[n]{x}}{x}$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-7** ***Derivada da Potência*** $x^{\frac{m}{n}}$

> *Seja* $x$ *a função identidade sobre reais,* $id : \mathcal{R} \to \mathcal{R}$, $m, n \in \mathcal{I}$, $n > 0$ *e* $m \neq 0$. *Se*
>
> *i)* $n$ *é par e* $x > 0$, *então* $(x^m)^{1/n} \equiv x^{m/n}$ *é diferenciável e a sua derivada dada por*
>
> $$D_x x^{\frac{m}{n}} = \frac{m}{n} x^{\frac{m}{n}-1} \equiv \frac{m}{n} \frac{(x^m)^{\frac{1}{n}}}{x}$$
>
> *ii)* $n$ *é ímpar e* $x \neq 0$, *então* $\sqrt[n]{x^m} \equiv (x^m)^{\frac{1}{n}}$ *é diferenciável e a sua derivada dada por*
>
> $$D_x \sqrt[n]{x^m} = \frac{m}{n} \frac{(x^m)^{\frac{1}{n}}}{x}$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-8** ***Derivada da Potência*** $x^r$

> *Seja* $x$ *a função identidade sobre reais,* $id : \mathcal{R} \to \mathcal{R}$, $r \in \mathcal{R}$ *e* $x > 0$. *Então* $x^r$ *é diferenciável e a sua derivada dada por*
>
> $$D_x x^r = r \cdot x^{r-1}$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-9** ***Derivada do Logaritmo Natural***

> *Seja* $x \in \mathcal{R}$ *e* $x > 0$. *Então* $\log x$ *é diferenciável e a sua derivada dada por*
>
> $$D_x \log x = \frac{1}{x}$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-10** ***Derivada da Exponencial Natural***

> *Seja* $x \in \mathcal{R}$. *Então* $e^x$ *é diferenciável e a sua derivada por*
>
> $$D_x e^x = e^x$$

A demonstração pode ser encontrada em [BEL66].

**Teorema B.3-11** ***Derivada de Exponencial de Base a***

> *Seja* $x \in \mathcal{R}$ *e* $a > 0$. *Então* $a^x$ *é diferenciável e a sua derivada por*
>
> $$D_x a^x = a^x \log_e a$$

# Apêndice C TESTES

## C.1 Testes

Esta secção apresenta um conjunto de exemplos que pretende ser representativo da generalidade dos programas do secundário. O símbolo $\rightarrow^{+}$ representa o fecho transitivo da relação $\rightarrow$ e o símbolo $\rightarrow^{0}$, a relação identidade (reflexividade) [BAA98]. Para todos estes exemplos, os testes efectuados demonstram a completude da base axiomática apresentada nesta tese.

### C.1.1 Sinal

**T1.** $-(-2) \rightarrow^{+} 2$

**T2.** $-(-(-2)) \rightarrow^{+} -2$

### C.1.2 Somas

**T1.** $1+3 \rightarrow 4$

**T2.** $-1+3 \rightarrow^{+} 2$

**T3.** $-\sqrt{x}+2\sqrt{x} \rightarrow^{+} \sqrt{x}$

**T4.** $-\sqrt{x}+(-2\sqrt{x}) \rightarrow^{+} -3\sqrt{x}$

**T5.** $((1+2)+3)+4 \rightarrow^{+} 10$

**T6.** $(((2+0)+(-1+3))+(4+(-1)))+(1+5) \rightarrow^{+} 13$

**T7.** $((2-4)+2)+(3-(3+4)) \rightarrow^{+} -4$

**T8.** $(2(4)+3)+2(5) \rightarrow^{+} 21$

**T9.** $\left(\left(\frac{1}{2}+\frac{1}{3}\right)+\frac{1}{4}\right)+\frac{1}{5} \rightarrow^{+} \frac{77}{60}$

**T10.** $3+(2^3-(5+\sqrt{4})) \rightarrow^{+} 4$

**T11.** $2(\sqrt{x}+1)+3(2\sqrt{x}+2) \rightarrow^{+} 8(\sqrt{x}+1)$

**T12.** $3\sqrt{x^3}+x\sqrt{x} \rightarrow^{+} 4x\sqrt{x}$

**T13.** $\frac{3}{5}\frac{\frac{3}{6}}{\frac{2}{4}}+\frac{5}{4} \rightarrow^{+} \frac{37}{20}$

**T14.** $\sqrt{x}+0 \rightarrow \sqrt{x}$

**T15.** $\sqrt{x}+\sqrt{x} \rightarrow 2\sqrt{x}$

**T16.** $\log_e(2x)+\log_e(3x) \rightarrow^{+} 2\log_e x+\log_e 6$

**T17.** $\log_e(\sqrt{4}\sqrt[3]{x})+\log_e(3x^2) \rightarrow^{+} \frac{7}{3}\log_e x+\log_e 6$

**T18.** $\frac{\mathrm{d}^2}{\mathrm{dx}^2}x^5+\frac{\mathrm{d}^2}{\mathrm{dx}^2}x^4 \rightarrow^{+} 20x^3+12x^2$

**T19.** $\frac{\mathrm{d}}{\mathrm{dx}}\log_e x^3+\frac{\mathrm{d}}{\mathrm{dx}}\sqrt{x}\rightarrow^+ 3\frac{1}{x}+\frac{1}{2}\frac{1}{\sqrt{x}}$

**T20.** $\frac{\mathrm{d}}{\mathrm{dx}}\log_e x^3+\frac{\mathrm{d}}{\mathrm{dx}}\log_e x\rightarrow^+ 4\frac{1}{x}$

**T21.** $\log_{10}5^3+\log_{10}4^2\rightarrow^+ \log_{10}2+3$

**T22.** $\log_{10}5^3+4\log_{10}2\rightarrow^+ \log_{10}2+3$

### C.1.3 Diferenças

**T1.** $x-(-4)\rightarrow x+4$

**T2.** $(x^2+1)-x^2\rightarrow^+ 1$

**T3.** $((1-2)-3)-4\rightarrow^+ -8$

**T4.** $(2-2x)-(x-3x)\rightarrow^+ 2$

**T5.** $\left(\left(\frac{1}{2}-\frac{1}{3}\right)-\frac{1}{4}\right)-\frac{1}{5}\rightarrow^+ -\frac{17}{60}$

**T6.** $4(x+1)-4x\rightarrow^+ 4$

**T7.** $\frac{\mathrm{d}}{\mathrm{dx}}x^2-\frac{\mathrm{d}}{\mathrm{dx}}\sqrt{x}\rightarrow^+ 2x-\frac{1}{2}\frac{1}{\sqrt{x}}$

### C.1.4 Produtos

**T1.** $4(-x)\rightarrow -4x$

**T2.** $4(x+1)\rightarrow^0 4(x+1)$

**T3.** $\left(\frac{1}{2}\sqrt{x}+2x\right)\frac{1}{3\sqrt{x}} \to^{+} \frac{1}{3}\left(2\sqrt{x}+\frac{1}{2}\right)$

**T4.** $\left(\sqrt{x}+2x^2\right)\frac{1}{\sqrt{x}} \to^{+} 2\left(x\sqrt{x}+\frac{1}{2}\right)$

**T5.** $2xxx \to^{+} 2x^3$

**T6.** $2xx(3x) \to^{+} 6x^3$

**T7.** $2\frac{1}{5}\frac{1}{6}\frac{1}{4}x \to^{+} \frac{1}{60}x$

**T8.** $2(3(2(-x))) \to^{+} -12x$

**T9.** $2(3(2(-x)))(4(-3x)) \to^{+} 144x^2$

**T10.** $4\frac{1}{\sqrt{x}}\frac{\sqrt{x}}{3}2 \to^{+} \frac{8}{3}$

**T11.** $2\log_e x\sqrt{x}(3x) \to^{+} 6x((\log_e x)\sqrt{x})$

**T12.** $2\log_e(x(\sqrt{x}(3x))) \to^{+} 2\left(\frac{5}{2}\log_e x+\log_e 3\right)$

**T13.** $2\frac{3+\frac{1}{4}}{6} \to^{+} \frac{13}{12}$

**T14.** $\frac{1}{5}\frac{\frac{1}{5}+\frac{4}{8}}{\frac{4}{6}-\frac{4}{2}} \to^{+} -\frac{21}{200}$

**T15.** $\frac{5}{2}\frac{3}{3^2} \to^{+} \frac{5}{6}$

**T16.** $\frac{1}{6}\frac{1}{x^{\frac{15}{18}}} \to^{+} \frac{1}{6}\frac{1}{\sqrt[6]{x^5}}$

**T17.** $\frac{1}{2x}\sqrt{x} \to^{+} \frac{1}{2}\frac{1}{\sqrt{x}}$

**T18.** $\frac{1}{x}\frac{1}{\sqrt{x}} \to^{0} \frac{1}{x}\frac{1}{\sqrt{x}}$

**T19.** $\frac{1}{3}\frac{\sqrt[3]{x^2}x}{\sqrt[3]{x^2}} \to^{+} \frac{1}{3}x$

**T20.** $\frac{1}{\sqrt[3]{x}}\frac{4}{3\sqrt[3]{x^2}} \to^{+} \frac{4}{3}\frac{1}{x}$

**T21.** $\sqrt{x}(2\sqrt{x}) \to^{+} 2x$

**T22.** $2\log_e x^{\frac{1}{2}} \to^{+} \log_e x$

**T23.** $2\log_e \sqrt{x} \to^{+} \log_e x$

**T24.** $2x\log_e \sqrt{x^3} \to^{+} 3x\log_e x$

### C.1.5 Quocientes

**T1.** $\frac{14}{2} \to 7$

**T2.** $\dfrac{15}{18} \to^{+} \dfrac{5}{6}$

**T3.** $\dfrac{(x^2+5x)+6}{(x^2+4x)+4} \to^{+} \dfrac{x+3}{x+2}$

**T4.** $\dfrac{(4x^2+12x)+9}{2x+3} \to^{+} 2\left(x+\dfrac{3}{2}\right)$

**T5.** $\dfrac{(4x^2-12x)+9}{3(2x-3)} \to^{+} \dfrac{2}{3}\left(x-\dfrac{3}{2}\right)$

**T6.** $\dfrac{2x+3\sqrt{x}}{3\sqrt{x}} \to^{+} \dfrac{2}{3}\sqrt{x}+1$

**T7.** $\dfrac{2(\log_e x+1)-3\log_e x}{2(\log_e x+2)} \to^{+} -\dfrac{1}{2}\dfrac{\log_e x-2}{\log_e x+2}$

**T8.** $\dfrac{2(\log_e x+1)-3\log_e x}{2(\log_e x-2)} \to^{+} -\dfrac{1}{2}$

**T9.** $\dfrac{x^2\sqrt[3]{x}}{\sqrt[3]{x^2}(3x)} \to^{+} \dfrac{1}{3}\sqrt[3]{x^2}$

**T10.** $\dfrac{3}{3^2} \to^{+} \dfrac{1}{3}$

**T11.** $\dfrac{1}{x^{\frac{4}{8}}} \to^{+} \dfrac{1}{\sqrt{x}}$

**T12.** $\dfrac{1}{x^{\frac{10}{15}}} \to^{+} \dfrac{1}{\sqrt[3]{x^2}}$

**T13.** $\dfrac{x^{\frac{5}{6}}}{(x^2)^{\frac{1}{3}}} \to^{+} \sqrt[6]{x}$

**T14.** $\dfrac{2}{\sqrt{2}} \to^{+} \sqrt{2}$

**T15.** $\dfrac{\sqrt{x}}{\sqrt[3]{x}} \to^{+} \sqrt[6]{x}$

### C.1.6 Potências

**T1.** $16^2 \to 256$

**T2.** $x^{3-2} \to^{+} x$

**T3.** $(\log_e x)^{3-2} \to^{+} \log_e x$

**T4.** $x^{\frac{2}{3}} \to \sqrt[3]{x^2}$

**T5.** $x^{\frac{3}{2}} \to^{+} x\sqrt{x}$

**T6.** $x^{\frac{15}{18}} \to^{+} \sqrt[6]{x^5}$

**T7.** $x^{\frac{18}{15}} \to^{+} x\sqrt[5]{x}$

**T8.** $x^{\frac{18}{75}} \to^{+} \sqrt[25]{x^6}$

**T9.** $x^{\frac{75}{18}} \to^{+} x^4\sqrt[6]{x}$

**T10.** $(2x)^{\frac{1}{3}} \rightarrow^{+} \sqrt[3]{2}\sqrt[3]{x}$

**T11.** $(\sqrt[3]{x^2})^2 \rightarrow^{+} x\sqrt[3]{x}$

## C.1.7 Raízes

**T1.** $\sqrt{16} \rightarrow^{+} 4$

**T2.** $\sqrt{8} \rightarrow^{+} 2\sqrt{2}$

**T3.** $\sqrt{4\sqrt{x}} \rightarrow^{+} 2\sqrt[4]{x}$

**T4.** $\sqrt{\frac{1}{4}\frac{1}{\sqrt{x}}} \rightarrow^{+} \frac{1}{2}\frac{1}{\sqrt[4]{x}}$

**T5.** $\sqrt[3]{4^6} \rightarrow^{+} 16$

**T6.** $\sqrt{(x+1)^3} \rightarrow^{+} (x+1)\sqrt{x+1}$

**T7.** $\sqrt{(\log_e x)^{3-2}} \rightarrow^{+} \sqrt{\log_e x}$

**T8.** $\sqrt{\frac{x^2-4}{x+2}} \rightarrow^{+} \sqrt{x-2}$

**T9.** $\sqrt[3]{-40} \rightarrow^{+} -2\sqrt[3]{5}$

**T10.** $\sqrt{((x^2+3x)+3)+\sqrt{x+1}\sqrt{x+1}} \rightarrow^{+} x+2$

## C.1.8 Logaritmos

**T1.** $\log_e 4 \rightarrow^{+} 2\log_e 2$

**T2.** $\log_4 16 \rightarrow^+ 2$

**T3.** $\log_e(x\sqrt{x}) \rightarrow^+ \frac{3}{2}\log_e x$

**T4.** $\log_e \frac{2}{4} \rightarrow^+ -\log_e 2$

**T5.** $\log_e 4^2 \rightarrow^+ 4\log_e 2$

**T6.** $\log_e 4^{\frac{1}{2}} \rightarrow^+ \log_e 2$

**T7.** $\log_e x^{\frac{1}{2}} \rightarrow \frac{1}{2}\log_e x$

**T8.** $\log_e(\sqrt{x})^3 \rightarrow^+ \frac{3}{2}\log_e x$

**T9.** $\log_e \sqrt{4} \rightarrow^+ \log_e 2$

**T10.** $\log_e \sqrt{x^3} \rightarrow^+ \frac{3}{2}\log_e x$

**T11.** $\log_{10}(5^3 2^4) \rightarrow^+ \log_{10} 2 + 3$

**T12.** $\log_{10} 2000 \rightarrow^+ \log_{10} 2 + 3$

**T13.** $\log_e(2e^{2x}) \rightarrow^+ 2x + \log_e 2$

### C.1.9 Exponenciais

**T1.** $e^{3+4} \rightarrow e^7$

**T2.** $\left(\frac{2}{4}\right)^{\log_e x} \to^+ \frac{1}{x}$

## C.1.10 Derivadas

**T1.** $\frac{\mathrm{d}}{\mathrm{dx}}(x+4) \to^+ 1$

**T2.** $\frac{\mathrm{d}}{\mathrm{dx}}(x+x) \to^+ 2$

**T3.** $\frac{\mathrm{d}}{\mathrm{dx}}(\log_e \sqrt[3]{x} + \log_e \sqrt{x}) \to^+ \frac{5}{6}\frac{1}{x}$

**T4.** $\frac{\mathrm{d}}{\mathrm{dx}}(\log_e x^2 - 2\log_e x) \to^+ 0$

**T5.** $\frac{\mathrm{d}}{\mathrm{dx}}\left(\frac{1}{2}\frac{1}{x\sqrt{x}}\right) \to^+ -\frac{3}{4}\frac{1}{x^2\sqrt{x}}$

**T6.** $\frac{\mathrm{d}}{\mathrm{dx}}(x^3\sqrt{x}) \to^+ \frac{7}{2}x^2\sqrt{x}$

**T7.** $\frac{\mathrm{d}^2}{\mathrm{dx}^2}\left(e^{\frac{x}{2}}e^{\frac{x}{4}}\right) \to^+ \frac{9}{16}e^{\frac{3}{4}x}$

**T8.** $\frac{\mathrm{d}}{\mathrm{dx}}\frac{\sqrt[3]{x}+\sqrt{x}}{x^2\sqrt[3]{x^2}} \to^+ -\left(\frac{5}{3}\frac{1}{x^2\sqrt[3]{x^2}} + \frac{3}{2}\frac{1}{x^2\sqrt[3]{x}}\right)$

**T9.** $\frac{\mathrm{d}^2}{\mathrm{dx}^2}\frac{\frac{1}{\sqrt[3]{x^2}}}{\sqrt{x}} \to^+ \frac{91}{36}\frac{1}{x^3\sqrt[6]{x}}$

**T10.** $\dfrac{\mathrm{d}}{\mathrm{dx}}\dfrac{x+1}{x-1} \to^{+} -2\dfrac{1}{(x-1)^2}$

**T11.** $\dfrac{\mathrm{d}}{\mathrm{dx}}\dfrac{\sqrt{x}}{x^2} \to^{+} -\dfrac{3}{2}\dfrac{1}{x^2\sqrt{x}}$

**T12.** $\dfrac{\mathrm{d}}{\mathrm{dx}}\dfrac{\sqrt{x}}{\sqrt[3]{x}} \to^{+} \dfrac{1}{6}\dfrac{1}{\sqrt[6]{x^5}}$

**T13.** $\dfrac{\mathrm{d}^3}{\mathrm{dx}^3}\dfrac{x\sqrt[3]{x^2}}{\dfrac{x^2\sqrt{x}}{\sqrt[3]{x}}} \to^{+} -\dfrac{15}{8}\dfrac{1}{x^3}\dfrac{1}{\sqrt{x}}$

**T14.** $\dfrac{\mathrm{d}^3}{\mathrm{dx}^3}\dfrac{x^2}{x^3} \to^{+} -6\dfrac{1}{x^4}$

**T15.** $\dfrac{\mathrm{d}^3}{\mathrm{dx}^3}\dfrac{x^2}{\sqrt{x}} \to^{+} -\dfrac{3}{8}\dfrac{1}{x}\dfrac{1}{\sqrt{x}}$

**T16.** $\dfrac{\mathrm{d}^2}{\mathrm{dx}^2}\dfrac{\sqrt{x}}{\sqrt[3]{x}} \to^{+} -\dfrac{5}{36}\dfrac{1}{x\sqrt[6]{x^5}}$

**T17.** $\dfrac{\mathrm{d}}{\mathrm{dx}}\dfrac{\log_e\sqrt{x}}{\sqrt[3]{x}} \to^{+} -\dfrac{1}{6}(\log_e x-1)\dfrac{1}{x\sqrt[3]{x}}$

**T18.** $\dfrac{\mathrm{d}^2}{\mathrm{dx}^2}\dfrac{\log_e\sqrt[3]{x}}{\sqrt{x}} \to^{+} \dfrac{1}{4}(\log_e x-2)\dfrac{1}{x^2\sqrt{x}}$

**T19.** $\dfrac{\mathrm{d}}{\mathrm{dx}}\sqrt{x} \to^{+} \dfrac{1}{2}\dfrac{1}{\sqrt{x}}$

**T20.** $\dfrac{\mathrm{d}}{\mathrm{dx}}\sqrt{\sqrt{x}+3\sqrt{x}} \to^{+} \dfrac{1}{2}\dfrac{1}{\sqrt[4]{x^3}}$

**T21.** $\dfrac{\mathrm{d}}{\mathrm{dx}}(\log_e(2x)+\log_e\sqrt{x}) \to^{+} \dfrac{3}{2}\dfrac{1}{x}$

**T22.** $\dfrac{\mathrm{d}}{\mathrm{dx}}\log_5 x \to^{+} \dfrac{1}{\log_e 5}\dfrac{1}{x}$

**T23.** $\dfrac{\mathrm{d}^2}{\mathrm{dx}^2}\log_e(x^3\sqrt{x}) \to^{+} -\dfrac{7}{2}\dfrac{1}{x^2}$

**T24.** $\dfrac{\mathrm{d}}{\mathrm{dx}}\log_e(4(x\sqrt{x^3})) \to^{+} \dfrac{5}{2}\dfrac{1}{x}$

**T25.** $\dfrac{\mathrm{d}^2}{\mathrm{dx}^2}(2\log_e\sqrt{x}+\log_e x^2) \to^{+} -3\dfrac{1}{x^2}$

**T26.** $\dfrac{\mathrm{d}}{\mathrm{dx}}4^x \to^{+} 2\log_e 2(4^x)$

# BIBLIOGRAFIA


[ALL88] Allen F. B. (1988). *Language and the Learning of Mathematics*. A speech delivered at the NCTM Annual Meeting, Chicago, April 1988 by Frank B. Allen, Emeritus Professor of Mathematics, Elmhurst College.

[AYR65] Ayres Jr, Frank (1965). *(Shaun's Outline of) Modern Abstract Algebra*. McGraw-Hill.

[AND95] Anderson J. R., Corbett A. T, Koedinger K. R. and Pelletier R. (1995). *Cognitive Tutors: Lessons Learned*. In The Journal of the Learning Sciences, 4 (2), pp. 167-207.

[BAA98] Baader F., Nipkow T. (1998). *Term Rewriting and All That*. Cambridge University Press, UK.

[BAA01] Baader F. and Snyder W. (2001). *Unification theory*. In A. Robinson and A. Voronkov, (Eds), Handbook of Automated Reasoning, Chap. 8, Elsevier Science.

[BAR00] Barwise J. and Etchemendy J. (2000). *Language Proof and Logic*. CSLI Publications, Leland Stanford Junior University. ISBN 1-889119-08-3.

[BEE03] Beeson M. (2003). *Mechanization of Mathematics*, Internet paper.

[BEE98] Beeson M. (1998). *Design Principles of Mathpert: Software to Support Education in Algebra and Calculus*. In Computer-Human Interaction in Symbolic Computation by Norbert Kajler (Ed.) Series

"Texts and Monographs in Symbolic Computation", 89-114. Springer-Verlag, Wien, New-York. ISBN: 3-211-82843-5.

[BEL66] Bell S., Blum J. R., Lewis J. V. (1966). *Modern University Calculus with Coordinate Geometry*. Holden-Day, Inc. San Francisco, California, USA.

[BER02] Bernardin L., McCarron J., and Harder D. (2002). *MathML in Maple*. MathML Conference 2002.

[BOR02] Borovansky P., Kirchner C., Kirchner H., Ringeissen C. (2002). *ELAN from a rewriting logic point of view*. Theoretical Computer Science 285(2), pp. 155-185.

[BUC96] Buchberger B. (1996). *Mathematica as a Rewrite Language*. In Ida T., Ohori A. and Takeichi M. (Eds.) Proceedings of the 2nd Fuji International Workshop on Functional and Logic Programming, 1-13. World Scientific, Singapore - New Jersey - London – Hongkong. (ISBN 9810229232) Copyright: World Scientific.

[BUN91] Bundy A. (1991). *A Science of Reasoning*. In J.-L. Lassez and G. Plotkin, (Eds), Computational Logic: Essays in Honor of Alan Robinson, 178-198. MIT Press.

[BUN85] Bundy A. (1985). *The Computer Modelling of Mathematical Reasoning*. Academic Press.

[BUN99] Bundy A. (1999). *A Survey of Automated Deduction*. Informatics Research Report EDI-INF-RR-0001. In Wooldridge M. J. and Veloso M. (Eds.), Artificial Intelligence Today: Recent Trends and Developments, LNAI Series 1600, 153-174. Springer-Verlag.

# Bibliografia


[CAV70] Caviness B. F. (1970). *On Canonical Forms and Simplification*. Journal of the ACM (JACM), v.17 n.2, pp. 385-396

[CHA00] Chan K. F. and Yeung D. Y. (2000). *Mathematical Expression Recognition: A Survey*. Technical Report HKUST-CS99-04, Department of Computer Science, Hong Kong University of Science and Technology, 1999. In Internacional Journal on Document Analyisis and Recognition (IJDAR), 3:1, 3-15. Springer-Verlag.

[CHA01] Chan K. F. and Yeung D. Y. (2001). *PenCalc: A Novel Application of On-Line Mathematical Expression Recognition Technology*. Proceedings of the Sixth International Conference on Document Analysis and Recognition, 774-778, Seattle, Washington, USA, 10-13 September 2001.

[CLO84] Clocksin W. F. and Mellish C. S. (1984). *Programming in Prolog Using the ISO Standard*. New York, Springer-Verlag.

[COH03] Cohen, J., S. (2003). *Computer Algebra and Symbolic Computation – Mathematical Methods*. A K Peters, Natick, Massachusetts, USA.

[COH02] Cohen, J., S. (2002). *Computer Algebra and Symbolic Computation – Elementary Algorythms*. A K Peters, Natick, Massachusetts, USA.

[DER90] Dershowitz N. and Jouannaud J. P. (1990). *Rewrite Systems*. In Handbook of Theoretical Computer Science, J. van Leeuwen, editor, Volume B, 243-320. Elsevier .

[DER01] Dershowitz N. and Plaisted D. A. (2001). *Rewriting*. In Robinson A. and Voronkov A. (Eds.) Handbook of Automated Reasoning. Elsevier Science Publishers, B.V.

[FAT92] Fateman, R. J. (1992). *A Review of Mathematica*. In Journal of Symbolic Computation (JSC), 13:5, 545-579. Elsevier.

[FAT94] Fateman R. J. (1994). *Symbolic Mathematics System Evaluators (extended abstract)*. In Y. N. Lakshman (ed.), Proc 1996 Intl. Symp. on Symbolic and Algebraic Computation, 86-94, New York. ACM Press.

[KAJ98] Kajler N. and Soiffer N. (1998). *A Survey of User Interfaces for Computer Algebra Systems*. Journal Of Symbolic Computation, 25(2):127-159. Elsevier, 1998(2).

[HAR02] Harris J. (2002). *MathML in Mathematica*. MathML Conference 2002.

[LUG93] Luger G. F. and Stubblefield W. A. (1993). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. California, The Benjamin/Cummings Publishing Company, Inc.

[MAR01] Martí-Oliet N. and Meseguer J. (2001). *Rewriting Logic - Roadmap and Bibliography*. Preprint submitted to Theoritical Computer Science.

[MAT96] Matos J. M. e Serrazina M. L. (1996). *Didáctica da Matemática*. Universidade Aberta, Portugal. ISBN: 972-674-172-6.

[MCA93] McArthur D., Lewis M., Bishay M. (1993). *The Roles of Artificial Intelligence in Education: Current Progress and Future Prospects*.

DRU-472-NSF. RAND, Santa Monica, CA.

[MES98] Meseguer J. (1998). *Research Directions in Rewriting Logic*. In Computational Logic, Berger U. and Schwichtenberg H. (Eds), NATO Advanced Study Institute, Marktoberdorf, Germany. Springer-Verlag.

[MOS71] Moses J. (1971). *Algebraic Simplification: A Guide for the perplexed*. Proceedings of the 2nd ACM symposium on Symbolic and Algebraic Manipulation, pp. 282-384, ACM.

[NAC01] Naciri H. and Rideau L. (2001). *The Marriage of MathML and Theorem Proving*. Proceedings Internet Accessible Mathematical Computation: A Workshop at ISSAC'2001.

[PER87] Pereira F. C. N., Shieber S. M. (1987). *Prolog and Natural Language Analysis*. Center for the Study of Language and Information, Stanford, CA. Digital Edition.

[RAV95] Ravaglia R. (1995). *Design Issues in a Stand Alone Multimedia Computer-based Mathematics Curriculum*. Education Program for Gifted Youth (EPGY). Research paper. Stanford University.

[RAV98] Ravaglia R., Alper T., Rozenfeld M., and Suppes P. (1998). *Successful Pedagogical Applications of Symbolic Computation*. In Computer-Human Interaction in Symbolic Computation by Norbert Kajler (Ed.) Series "Texts and Monographs in Symbolic Computation", 61-87. Springer-Verlag, Wien, New-York. ISBN: 3-211-82843-5.

[RUS95] Russell S. and Norvig P. (1995). *Artificial Intelligence: A Modern*

*Approach*. New Jersey, Prentice Hall.

[RIC91] Rich E. and Knight K. (1991). *Artificial Intelligence*. International Edition, McGraw Hill Inc.

[SOL82] Solow D. (1982). *How to Read and Do Proofs: An Introduction to Mathematical Thought Process*. John Wiley &Sons, New York.

[STE00] Sterling L. and Shapiro E. (2000). *The Art of Prolog*. London, England, The MIT Press.

[TRA93] Trafton J. G. and Reiser B. J. (1993). *The Contributions of Studying Examples and Solving Problems*. In Polson M. C. (Ed.) Proceedings of the 15th Annual Conference of the Cognitive Science Society. Hillsdale, NJ, Lawrence Erlbaum.

[WES99] Wester M. (1999). *A Critique of the Mathematical Abilities of CA Systems*. In Wester M. J. (Ed.) Computer Algebra Systems: A Practical Guide. John Wiley & Sons, Chichester, UK. ISBN 0-471-98353-5, xvi+436 pages.

[VAL00] Valença J. M. E. e Barros J. B. (2000). *Fundamentos de Computação – Livro I: Computação e Linguagem*. Universidade Aberta, Portugal. ISBN: 972-674-317-6.

# ERRATA

## CAPÍTULO 1

| | | |
|---|---|---|
| **Secção 1.1, pág. 4** | *Onde se lê* | … tanto pode ser irredutível (normal) como irredutível (redutora ou potencialmente redutora). |
| | *Deve-se ler* | … tanto pode ser irredutível (normal) como redutível (redutora ou potencialmente redutora). |
| **Secção 1.2, pág. 9** | *Onde se lê* | No Capítulo 1 é abordado o trabalho relacionado e a base teórica … |
| | *Deve-se ler* | No Capítulo 3 é abordado o trabalho relacionado e a base teórica … |
| **Secção 1.2, pág. 10** | *Onde se lê* | No Capítulo 1 é discutido o Assistente de Matemática que desenvolvemos … |
| | *Deve-se ler* | No Capítulo 5 é discutido o Assistente de Matemática que desenvolvemos … |

## CAPÍTULO 2

| | | |
|---|---|---|
| **Secção 2.2,** | *Onde* | $\mathcal{K} = \{ f(s,t) \mid f \in \mathcal{O}p \wedge s,t \in \mathcal{T} \setminus \{x\} \}$ |

| | | |
|---|---|---|
| **pág. 17, i)** | *se lê* | |
| | *Deve-se ler* | $\mathcal{K} = \{ f(s,t) \mid f \in \mathcal{O}p \wedge s,t \in \mathcal{T}_{\mathcal{A}} \setminus \{x\} \vee s,t \in \mathcal{K} \}$ |
| **Secção 2.2, pág. 17, ii)** | *Onde se lê* | $\mathcal{F} = \{ f(s,t) \mid f \in \mathcal{O}p \wedge s \in \{x\} \wedge t \in \mathcal{T} \vee s \in \mathcal{T} \wedge t \in \{x\} \}$ |
| | *Deve-se ler* | $\mathcal{F} = \{ f(s,t) \mid f \in \mathcal{O}p \wedge s \in \{x\} \vee t \in \{x\} \vee s,t \in \mathcal{F} \}$ |
| **Secção 2.2, pág. 19** | *Onde se lê* | $\mathcal{I}^{-} = \{ f(s,t) \in \mathcal{T} \mid f \in \{prod\} \wedge s \in \mathcal{S}^{-} \wedge t \in \mathbb{N} \}$ |
| | *Deve-se ler* | $\mathcal{I}^{-} = \{ f(s,t) \in \mathcal{T} \mid f \in \{prod\} \wedge s \in \mathcal{S}^{-} \wedge t \in \mathbb{N} \setminus \{0\} \}$ |
| **Secção 2.2, pág. 22** | *Onde se lê* | $\mathcal{S}^{-} = \{ f(s,t) \in \mathcal{T} \mid f \in \{prod\} \wedge s \in \mathcal{S}^{-} \wedge t \in \{1\} \}$ |
| | *Deve-se ler* | $\mathcal{S}^{-} = \{ f(s,t) \in \mathcal{T} \mid f \in \{prod\} \wedge (s \in \{-1\} \vee s \in \mathcal{S}^{-}) \wedge t \in \{1\} \}$ |
| **Secção 2.4.2, pág. 52** | *Onde se lê* | De facto, o nível de uma estrutura determina o nível de componentes presentes nessa estrutura. |
| | *Deve-se ler* | De facto, o nível de uma estrutura determina o número de componentes presentes nessa estrutura. |
| **Secção 2.4.2, pág. 53** | *Onde se lê* | A expressão logaritmica possui apenas uma (1) componente, a expressão raiz, embora … |
| | *Deve-se ler* | A expressão logaritmica, para além da componente principal, possui apenas uma (1) componente, a expressão raiz, embora … |
| **Secção 2.4, pág. 59, ii)** | *Onde se lê* | $\mathcal{NR}edex(\Xi\text{-}a) \geq \mathcal{NR}edex(\Xi\text{-}b)$ |

| | | |
|---|---|---|
| | *Deve-se ler* | $\mathcal{NR}edex(\Xi\text{-}a) > \mathcal{NR}edex(\Xi\text{-}b)$ |

## CAPÍTULO 3

| | | |
|---|---|---|
| **Secção 3.3.2, pág. 98** | *Onde se lê* | $y = b$ |
| | *Deve-se ler* | $y = a$ |

## CAPÍTULO 4

| | | |
|---|---|---|
| **Secção 4.2, pág. 117** | *Onde se lê* | $c \wedge t \in \mathcal{K}^+ \setminus \{-1,0\} \wedge t \in \mathcal{K}^+ \setminus \{-1,0,1\} \wedge$ |
| | *Deve-se ler* | $c \wedge t \in \mathcal{K}^+ \setminus \{0\} \wedge t \in \mathcal{K}^+ \setminus \{0,1\} \wedge$ |
| **Secção 4.3, pág. 135** | *Onde se lê* | Mas como veremos, a maioria das transformações lógicas deste tipo, com a excepção das regras de cálculo como, por exemplo, o cálculo de derivadas, são, reescritas … |
| | *Deve-se ler* | Mas como veremos, a maioria das transformações lógicas deste tipo são reescritas … |

| | | |
|---|---|---|
| **Secção 4.3, pág. 168** | *Onde se lê* | $\mathcal{C}_2^{\psi} = \left\{ f(s, g(t,u)) \in \mathcal{T} \mid f\_g \in \mathcal{S}ig_2^{\psi} \wedge s,t \in \mathcal{T} \right\}$ |
| | *Deve-se ler* | $\mathcal{C}_2^{\psi} = \left\{ f(s, g(t,u)) \in \mathcal{T} \mid f\_g \in \mathcal{S}ig_2^{\psi} \wedge s,t,u \in \mathcal{T} \right\}$ |
| **Secção 4.3, págs. 178 e 179** | *Onde se lê* | ***if*** $s \in \mathbb{N} \bigcup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{0,1\} \wedge r \in \mathcal{T}^+ \bigcup \{0,1\} \wedge$ |
| | *Deve-se ler* | ***if*** $s \in \mathbb{N} \bigcup \{e\} \setminus \{0,1\} \wedge t \in \mathbb{N} \setminus \{0,1\} \wedge r \in \mathcal{T}^+ \setminus \{0,1\} \wedge$ |
| **Secção 4.4, pág. 193** | *Onde se lê* | Por exemplo, no caso da expressão, uma … |
| | *Deve-se ler* | Por exemplo, no caso da expressão $((1+2)+3)+4$, uma … |
| **Secção 4.4, pág. 210** | *Onde se lê* | As estratégias estritas são, por outro lado, estratégias que tendem a reduzir ou, quanto muito, manter o grau de complexidade … |
| | *Deve-se ler* | As estratégias estritas são, por outro lado, estratégias que tendem a reduzir o grau de complexidade … |
| **Secção 4.4, pág. 218** | *Onde se lê* | $\log_e 4^2 \rightarrow_{\alpha} 2\log_e 4 \downarrow^1 \rightarrow_{\gamma} 2\log_e 2^2 \rightarrow_{\beta}$ |
| | *Deve-se ler* | $\log_e 4^2 \rightarrow_{\alpha} 2\log_e 4 \downarrow^1 \rightarrow_{\gamma} 2\log_e 2^2 \rightarrow_{\beta}$ $\rightarrow_{\beta} \log_e (2^2)^2 \downarrow^2 \not\rightarrow_{\kappa} \log_e 4^2$ |
| **Secção 4.4, pág. 222** | *Onde se lê* | Assim, se existir mais do que uma solução, as soluções devem interceptar uma outra, pelo menos, uma vez. |
| | *Deve-se ler* | Assim, se existir mais do que uma solução, as soluções devem interceptar, pelo menos, uma outra. |

## CAPÍTULO 5

| | | |
|---|---|---|
| **Secção 5.3, pág. 236** | *Onde se lê* | *As suas estruturas são, porém, de níveis diferentes.* |
| | *Deve-se ler* | *As suas estruturas são do mesmo nível.* |

## CAPÍTULO 6

| | | |
|---|---|---|
| **Secção 6.1, pág. 274** | *Onde se lê* | *O resultado de reescrita é uma expressão mais simples ou uma expressão que se pode tornar mais num número finito de passos.* |
| | *Deve-se ler* | *O resultado de reescrita é uma expressão mais simples ou uma expressão que se pode tornar mais simples num número finito de passos.* |