Projeto Final em Engenharia Informática

UAbALL – Automata Learning Lab

SIMULAÇÃO DE AUTÓMATOS FINITOS DETERMINISTAS (DFA)

Autor
André Maciel da Silva e Sousa | 1300012 | andremaciel.pt
Orientador
Professor António Jorge do Nascimento Morais

Porto, 14 de setembro de 2020

Agradecimentos

Sendo este manuscrito o culminar da licenciatura em Engenharia Informática, num caminho longo, iniciado no ano 2013, é momento de agradecer a todos os que colaboram neste percurso.

Aos meus pais que me ensinaram a força do amor e dos valores.

À minha esposa, Marta Silva, que sempre esteve presente e me ajuda a ser melhor pessoa.

Aos meus filhos Alícia e Tomás, com quem aprendi o amor incondicional.

Aos meus amigos António Páscoa e Fernando Teixeira Pinto, por me terem colocado este desafio formativo.

Aos meus amigos e administradores do Grupo Auto Soluções Mobilidade, Alfredo, Francisco e Pedro Barros Leite, pela disponibilidade, apoio e espírito critico.

Aos docentes da Licenciatura em Engenharia Informática da Universidade Aberta, pelo apoio em todo o processo académico.

Aos colegas Andreia Romão, Cláudia Pêgo, José Manuel Sousa, Márcio Santos e Sergio Meren, que autorizaram a referência pelo seu apoio no inquérito do UAbALL.

Aos colegas Nelson Russa, Paulo Gomes e Tiago Candeias, e **aos professores** Vitor Rocio e Leonel Morgado pelo apoio e colaboração na criação do Clube de Informática da Universidade Aberta.

Ao meu orientador e amigo Jorge Morais, pela paixão transmitida pelos autómatos e pelo acompanhamento neste projeto com uma disponibilidade 24/7.

À Universidade Aberta cujo Modelo Pedagógico permitiu este percurso.

Conteúdo

Lista de Figuras

Lista de Tabelas

Resumo

O Automata Learning Lab da Universidade Aberta (**UAbALL**), pretende ser um laboratório integrado de simulação de autómatos. Numa primeira fase focado na construção da base e introduzindo a Simulação de Autómatos Finitos Deterministas (DFA). Este Laboratório ambiciona gozar de capacidade de extensibilidade e adaptabilidade, sendo este documento uma base técnica e científica para que no futuro sejam produzidas as restantes componentes, assim como adaptado a novas realidades tecnológicas e plataformas de distribuição.

Durante este trabalho são utilizados Conceitos Centrais da Teoria de Autómatos, tais como **Alfabeto** ($\sum$) – Conjunto finito de símbolos; **String** – Sequência finita de símbolos escolhidos de um alfabeto; **Cadeia Vazia**(ε) – ou *String* vazia; **Comprimento de *String***($|\omega|$) – numero de elementos na *String*; **Concatenação de *Strings*** ($x = 011; y = 100; xy = 011100$); **Linguagem($\mathcal{L}$)** – conjunto de símbolos que obedece a uma propriedade num Alfabeto; **Construtores de Conjuntos** – para definição de Linguagens, por exemplo $\{ \omega \mid \omega$ é $um\ numero\ binario\ primo \}$.

O Capítulo 1 consiste na apresentação formal do projeto, delineando os objetivos e apresentando o plano de trabalho.

Para o segundo Capítulo ficou reservado o processo de conceção, com respetivas especificações e desenhos, desde a aplicação dos idiomas, passando pelos menus, funcionalidades globais, interface gráfica, discussão e decisão do ambiente e definição de recursos.

O Capítulo 3, dedica-se à implementação, dos temas e abordagens discutidas no capítulo anterior, concluindo com os testes à aplicação e discussão dos resultados obtidos.

Antes da apresentação das conclusões, é disponibilizada a versão 0.1 do laboratório, e delineado o plano de acompanhamento de novas versões.

Todo o código resultante deste projeto, é anexado no final deste manual de projeto, que se pretende Relatório técnico para trabalho futuro.

Palavras-chave: Autómatos Finitos, Linguagens e Expressões Regulares, Linguagens e Gramáticas Independentes de Contexto, Máquinas de Turing.

Abstract

The "Automata Learning Lab da Universidade Aberta (UAbALL)" strives to be an integrated laboratory for simulation of Automata. A first phase focused on building the base and introducing the Simulation of Deterministic Finite Automata (DFA). This Laboratory intends to enjoy the capacity of extensibility and adaptability, being this document a technical and scientific basis for the future production of the remaining components, as well as adapted to new technological realities and distribution platforms.

During this work, Central Concepts of Automata Theory are used, such as **Alphabet** (∑) - Finite set of symbols; **String** - Finite sequence of symbols chosen from an alphabet; **Empty String** (ε) - or Empty String; **String Length** ($|\omega|$) - number of elements in the *String*; **String Concatenation** ($x = 011; y = 100; xy = 011100$); **Language** ($\mathcal{L}$) - set of symbols that obey a property in an Alphabet; **Set Builders** - for defining Languages, for example $\{ \omega \mid \omega \text{ is a prime binary number} \}$..

Chapter 1 consists of the formal presentation of the project, outlining the objectives and presenting the work plan.

For the second Chapter, the design process was reserved, with respective specifications and drawings, from the application of languages, through menus, global functionalities, graphical interface, discussion and decision of the environment, plus the definition of resources.

Chapter 3 is dedicated to the implementation of the themes and approaches addressed in the former chapter, concluding with tests on the application and discussion regarding the results achieved.

Before the presentation concerning the conclusions, version 0.1 of the laboratory is made available, and the plan for monitoring new versions is outlined.

All code resulting from this project is attached to the end of this project manual, which indicates the technical report for future work.

Keywords: Finite Automota, Regular Expressions and Languages, Context-Free Grammars and Languages, Turing Machines.

Capítulo 1

Descrição e objetivos do trabalho

Pretende-se iniciar a construção de um laboratório virtual integrado de simulação de autómatos, gramáticas e máquinas de *Turing*, que permitam dar apoio às unidades curriculares de Linguagens e Computação e de Compilação, bem como disponibilizar o seu acesso à comunidade académica em geral. Neste projeto, em particular, pretende-se iniciar a implementação do laboratório virtual, incluindo já a implementação da simulação de autómatos finitos deterministas.

1.1. PROPOSTA PARA O TRABALHO

A principal motivação para este projeto é a vontade de produzir conteúdo útil para Ensino a Distância (EaD), somada com a aplicação dos conhecimentos adquiridos, nas áreas que mais me senti identificado, ao longo da licenciatura em Engenharia Informática. Também existe uma vontade de prosseguir estudos em EaD, na materialização da digitalização de processos.

O projeto inicia-se com o planeamento do laboratório, sendo necessário decidir qual o ambiente ou ambientes em que fará sentido a sua implementação (*Website*, App para telemóvel e tablet, Aplicação em Java, etc...) e a implementação em, pelo menos, um destes, de uma versão inicial. Para finalizar, deve ser implementado um simulador de autómatos finitos deterministas, de modo a poder testar o mesmo no laboratório e fazer as melhorias que se considerem pertinentes.

Os recursos dependem do ambiente de implementação. Incluirão sempre programação numa linguagem com interface gráfica e acesso a uma base de dados.

1.2. OBJETIVOS

Laboratório Virtual: Operacionalizar a base do Laboratório Virtual UAbALL, como também o seu primeiro módulo: SIMULAÇÃO DE AUTÓMATOS FINITOS DETERMINISTAS (DFA).

Extensibilidade e adaptabilidade: A plataforma inicial deverá ser a base de todas as funcionalidades esperadas com este laboratório virtual integrado de simulação de autómatos, gramáticas e máquinas de *Turing*. Também poderá permitir em conformidade com informação da localização, ou escolha do cliente, alterar o idioma de apresentação.

Relatório técnico para trabalho futuro: Produzir informação técnica suficiente para que o projeto possa ser adaptado a novas realidades tecnológicas e plataformas de distribuição.

1.3. CRONOGRAMA

Identificação	Descrição	Data Limite
	R0. Proposta Inicial	01-abril-2020
A - Conceção	A1. Especificações do Projeto A1.1 – Idiomas A1.2 – Definição Menus A1.3 – Funcionalidades Globais A1.4 - Regras DFA A1.5 – Definições Interface Gráfica A1.6 – Decisão Ambiente A1.7 – Definição dos Recursos	15-abril-2020
	A2. Desenho das Especificações A2.1 – Tabela de Idiomas A2.2 – Modelo Menus A2.3 – Funcionalidades Globais A2.4 – Registo Regras DFA A2.5 – Componentes Interface Gráfica	29-abril-2020
	R1. Relatório Intermédio	06-maio-2020
B - Implementação	B1. Desenvolvimento B1.1 – Idiomas B1.2 – Menus B1.3 – Funcionalidades Globais B1.4 – Funcionalidades DFA B1.5 – Interface Gráfica	05-junho-2020
	B2. Testes	18-junho-2020
	R2. Relatório Final	24-junho-2020
C – Manutenção	C1. Disponibilização Versão 1.0	06-julho-2020
	C2. Release Fixes	16-julho-2020
	P0. Preparação da Defesa do Projeto	A validar [1]

Tabela 1 - Cronograma

[1] https://elearning.uab.pt/mod/forum/discuss.php?d=610240

1.4. MAPA DE GANTT

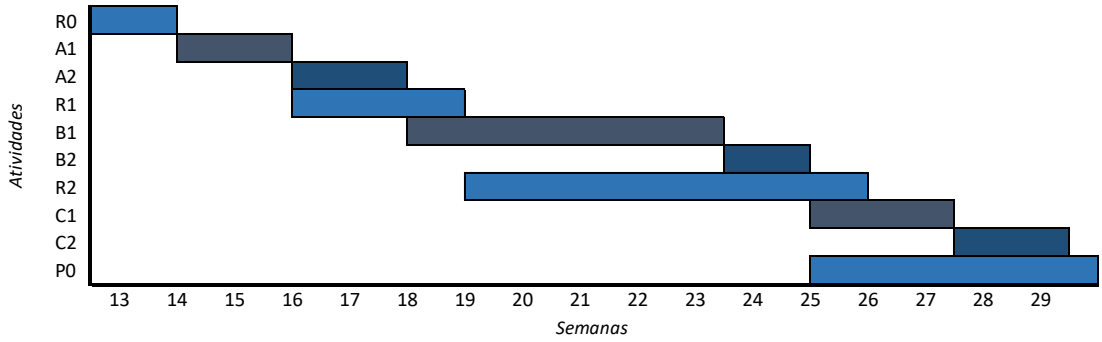
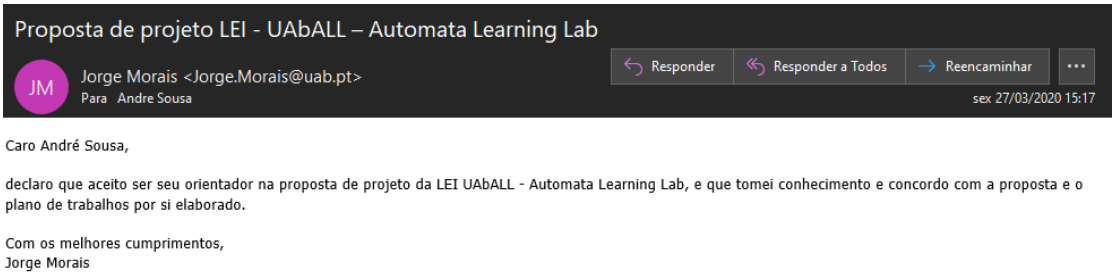


Figura 1 - Mapa de Gantt

1.5. ACEITAÇÃO DO PROFESSOR ORIENTADOR



Proposta de projeto LEI - UAbALL – Automata Learning Lab

JM Jorge Morais <Jorge.Morais@uab.pt>
Para Andre Sousa

Responder | Responder a Todos | Reencaminhar | ...
sex 27/03/2020 15:17

Caro André Sousa,

declaro que aceito ser seu orientador na proposta de projeto da LEI UAbALL - Automata Learning Lab, e que tomei conhecimento e concordo com a proposta e o plano de trabalhos por si elaborado.

Com os melhores cumprimentos,
Jorge Morais

Capítulo 2

A- Conceção

O projeto inicia-se com o planeamento do laboratório. Neste capítulo decidimos qual o ambiente ou ambientes em que fará sentido a sua implementação. Discutiremos, primeiramente as especificações para o cumprimento dos objetivos delineados, concluindo com o desenho dessas especificações.

A.1 - ESPECIFICAÇÕES DO PROJETO

Paras as especificações devemos tomar em consideração aqueles que foram os objetivos traçados para o projeto, nomeadamente o Laboratório Virtual no seu todo, o primeiro módulo (DFA), a sua extensibilidade e adaptabilidade, bem como informação técnica suficiente para a sua manutenção e distribuição.

A1.1 – Idiomas

Pretendendo um Laboratório Virtual universal, é necessário que esteja disponível em vários idiomas, melhorando a aprendizagem dos estudantes, suprimindo a adaptação linguística. A seleção do idioma inicial deverá acontecer, automaticamente e com base no idioma do navegador utilizado. Esta definição contraria a premissa inicial (com base na localização) por ser considerado que o idioma selecionado no navegador será o da preferência do utilizador, enquanto a localização acrescenta maior indefinição, por exemplo um estudante português em Erasmus em Inglaterra. Os idiomas base do projeto serão o português, castelhano e inglês, ficando outros para futuras expansões ao UAbALL. A figura 2 representa o fluxo de decisão para a opção do idioma a apresentar inicialmente ao utilizador final.

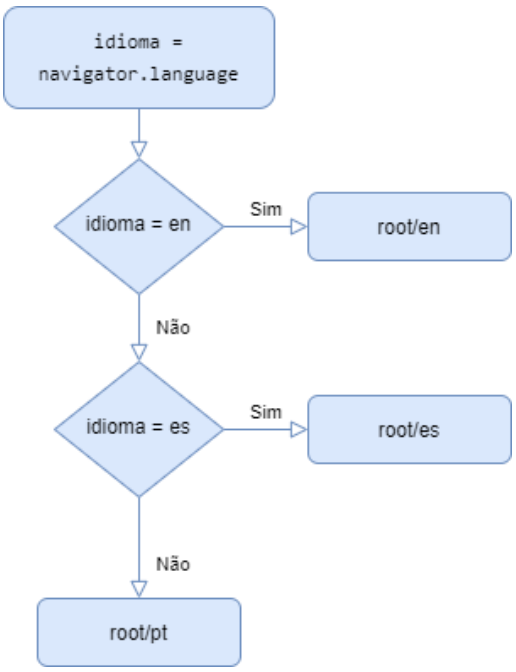


Figura 2 - Seleção Idioma Inicial

A1.2 – Definição Menus

Para o menu pretendemos um acesso simplificado e claro a todas as funcionalidades que estarão disponíveis numa versão mais avançada deste laboratório. Com base na Unidade Curricular de Linguagens e Computação são considerados as seguintes opções:

DFA – Autómatos Finitos Deterministas

NFA – Autómatos Finitos Não-Deterministas

NFA-ε – Autómatos Finitos Não-Deterministas com Movimentos Vazios

PDA – Autómato Pilha

TURING – Máquina de Turing

REGEX – Expressões Regulares

GRAMMAR – Gramática Livre de Contexto

A página "home" deverá conter uma breve descrição científica de cada uma das opções disponíveis, assim como para as possíveis relações e transformações.

A1.3 – Funcionalidades Globais

Pretendendo que o UAbALL seja um laboratório virtual integrado de simulação das linguagens formais e demais matérias abordadas nas unidades curriculares, referenciadas no primeiro capítulo, este deverá ser agilmente implementado na plataforma moodle, ou adaptável a qualquer plataforma de ensino digital. Como qualquer sistema distribuído é igualmente importante garantir transparência de funcionamento, disponibilidade e fiabilidade, mas também modelos uniformizados de utilização e adaptados aos processos de aprendizagem e ensino das linguagens formais e teoria de autómatos.

A1.4 – Regras DFA

As opções para o primeiro modulo acrescem de importância, pelo facto de influenciarem as opções para futuros módulos, pelo princípio de homogeneidade como alavanca de uma favorável adaptação ao laboratório, e deste modo reduzir os custos de aprendizagem ao utilizador final.

O modulo DFA deverá permitir contruir e validar um autómato desta natureza, validando que estamos perante um DFA, na secção A2.2.1 é apresentada a definição formal de DFA.

A1.5 – Definições Interface Gráfica

Para a interface gráfica deveremos tomar em conta todas as especificações do projeto, que pelas definições anteriores concluímos que estamos perante uma adoção em ambiente *web* com recurso aos navegadores, que acrescentará algum trabalho acrescido de garantia de funcionamento uniformizado, ou, não o sendo, que exista minimização das diferenças. Para a simulação do autómato, iremos necessitar de um suporte que aceite por todos os navegadores *web* modernos através de biblioteca JavaScript, pelo que a opção tomada passa pela biblioteca *Graphviz*[2] para *SVG - Scalable Vector Graphics*[3] para a apresentação dos grafos resultantes.

A1.6 – Decisão Ambiente

Após pesquisa aprofundada não foi encontrada nenhuma aplicação *web* que incluísse todas as características pretendidas para o UAbALL, com especial relevo para a expansibilidade e adaptabilidade. Inicialmente havíamos abordado a necessidade de decisão do ambiente ou ambientes em que faria sentido a implementação do UAbALL, tomando em consideração a multiplicidade de ecrãs utilizados pelos futuros utilizadores, somando as soluções técnicas atuais, a decisão recaí sobre a construção de um *Website*.

Utilizaremos o ***Responsive Web Design*** que oferece um conjunto de ferramentas que permitem criar páginas *web* que respondem aos diferentes tamanhos de ecrã. São utilizadas *grids* fluidas, imagens flexíveis e *media queries* para adaptar as páginas *web* independentemente das dimensões de ecrã do dispositivo (Sampaio, 2013).

Com esta decisão somamos ao UAbALL capacidade de estar presente e disponível em qualquer dispositivo e plataforma com acesso à *World Wide Web*.

A1.7 – Definição dos Recursos

O ambiente de desenvolvimento utilizado durante a realização deste projeto, foi o sistema operativo Microsoft Windows 10 Pro, com as seguintes ferramentas e recursos:

- Visual Studio Code Versão 1.44.2 - editor de código-fonte desenvolvido pela Microsoft para Windows, Linux e macOS.
- Para os testes de navegação *web,* foram utilizados os seguintes *browsers*:
    - Brave Versão 1.7.92
    - FireFox Versão 75.0
    - Google Chrome Versão 80.0.3987.163
    - Internet Explorer Versão 11.0.185
    - Microsoft Edge Versão 18363
    - Opera Versão 63.0.3368

[2] https://www.graphviz.org/
[3] https://www.w3.org/TR/SVG/

- HTML5 - *HyperText Markup Language* – linguagem de marcação para construção de páginas *web,* neste projeto é utilizada a quinta versão, que inclui novas funcionalidades que possibilita a utilização de novos recursos, como *API's*[4] para gráficos bidimensionais.
- CSS - *Cascading Style Sheets* – linguagem para adicionar estilos ao modo como são apresentados os documentos HTML.
- JavaScript – linguagem de programação interpretada de alto nível, que conjuntamente com HTML e CSS perfaz as três principais tecnologias da *World Wide Web,* permitindo a criação de páginas *web* interativas, e todos os principais navegadores têm um mecanismo *JavaScript* dedicado para executá-lo.
- Alojamento para testes online em servidor próprio[5].
- Moodle Versão 3.3.9+ da Universidade Aberta, no espaço do Clube de Informática[6], que será utilizado para testes e avaliação do projeto.
- jQuery - uma biblioteca JavaScript desenvolvida para simplificar os scripts interpretados no navegador do utilizador.
- Viz.js - um cliente de Graphviz[7] em JavaScript.

A2. DESENHO DAS ESPECIFICAÇÕES

Concluída a discussão das especificações para o cumprimento dos objetivos delineados, é chegado o momento do desenho destas. Nesta secção serão tratadas as especificações para as especificações dos Idiomas, Menu, Funcionalidades Globais, Regras e Funcionalidades para o modulo DFA e Componentes Gráficas a implementar nesta versão.

A2.1 – Tabela de Idiomas

Tomando em consideração que a plataforma será alojada num servidor *web*, exemplo www.uaball.com, para a implementação dos idiomas serão utilizados subdiretórios com nomenclatura em conformidade com a ISO 639.

---

[4] ***Application Programming Interface***, é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços.

[5] https://andremaciel.pt/UAb/UAbALL/

[6] https://elearning.uab.pt/mod/page/view.php?id=554247

[7] **Graphvizé** um software de visualização de grafos de código aberto. A visualização gráfica é uma forma de representar informações estruturais como diagramas de gráficos e redes abstratas. Possui aplicações importantes em redes, bioinformática, engenharia de software, banco de dados e web design, aprendizagem de máquinas e interfaces visuais para outros domínios técnicos.

Code	ISO 639-1 language name			Endonym	URI
	English	French	German		
en	English	anglais	Englisch	English	www.uaball.com/en
es	Spanish	espagnol	Spanisch	español	www.uaball.com/es
pt	Portuguese	portugais	Portugiesisch	português	www.uaball.com/pt

Tabela 2 - Exemplos ISO 639-1

O utilizador poderá alterar o idioma através da seleção da respetiva bandeira que estará colocada de modo visível no topo da página *web* aplicação.

A2.2 – Modelo Menus

O menu deverá incluir na sua navegabilidade as opções de idioma discutidas anteriormente, assim como uma opção "home", para regresso ao ecrã de boas vindas. A estas acrescem DFA para Autómatos Finitos Deterministas, NFA – Autómatos Finitos Não-Deterministas, NFA-ε – Autómatos Finitos Não-Deterministas com Movimentos Vazios, REGEX para Expressões Regulares, PDA para Autómato Pilha e GRAMMAR para Gramática Livre de Contexto e TURING – Máquina de Turing.

A2.2.1 – Definições básicas sobre a Teoria de Autómatos

As definições neste projeto são baseadas nas obras de (Menezes, 2000) e (Hopcroft, Motwani, & Ullman, 2007)[8].

Um **autómato finito** é um modelo matemático de uma máquina computacional de estados finitos que lê uma *String* e aceita ou rejeita essa entrada.

Este modelo é composto por uma **Fita**, que funciona como dispositivo de entrada que contem a *String* a ser processada por uma **Unidade de Controlo** que acede a uma célula da Fita de cada vez e movimenta-se para uma nova posição da Fita, **Função de Transição** que dado um estado atual da máquina e um símbolo lido, determina uma mudança de estado. O autómato muda de estado de acordo com a Função e para a execução, uma vez lida a *String*, emitindo mensagem de aceitação ou rejeição da sequência, se a paragem tiver ocorrido num estado pertencente ao conjunto de estados finais, ou rejeição, caso contrário. Pode ainda haver rejeição antes do fim da leitura da String no caso de, dado o estado atual e o símbolo lido, não haver transição prevista para o par (estado, símbolo)[9]. Um exemplo desta execução é representado na figura 3, para o reconhecimento de palavras terminadas em "*ing*".

[8] Definições adicionais podem ser encontrada nas referências citadas
[9] Esta situação também pode acontecer se o símbolo lido não pertencer ao alfabeto.

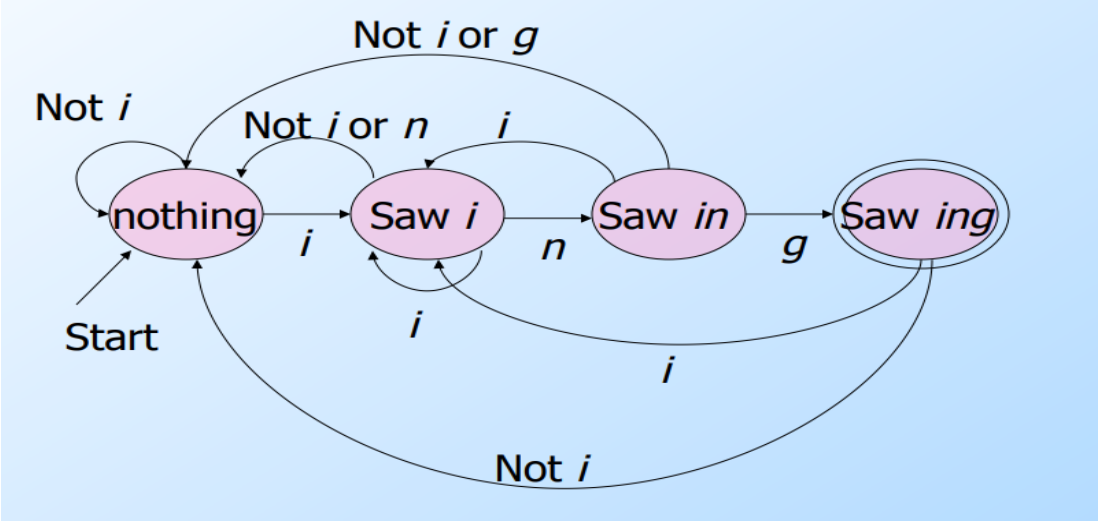


Figura 3 - Reconhecimento Strings terminadas em "ing"

Normalmente, os autómatos estão classificados nos seguintes grupos:

DFA – Autómatos Finitos Deterministas – apresentam uma Função em que para cada estado e cada símbolo, possui uma única transição, começando em um estado inicial e avança para próxima célula após leitura de um símbolo. É representado formalmente por uma 5-tupla (Q, Σ, δ, qo, F), onde:

- Q é um conjunto finito de estados.
- Σ é um conjunto finito de símbolos, Alfabeto.
- δ é a função de transição, isto é, δ: Q x Σ → Q.
- qo é o estado inicial, isto é, o estado do autómato antes de qualquer entrada ser processada, onde qo ∈ Q.
- F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.

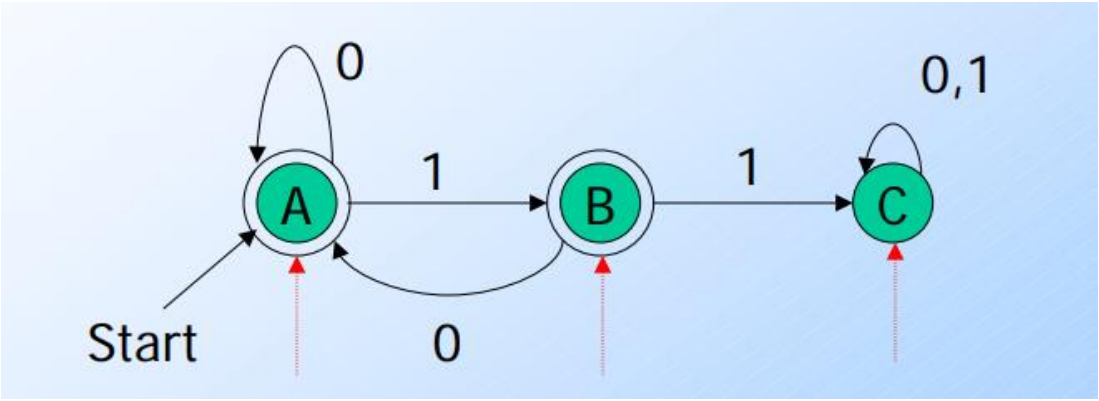
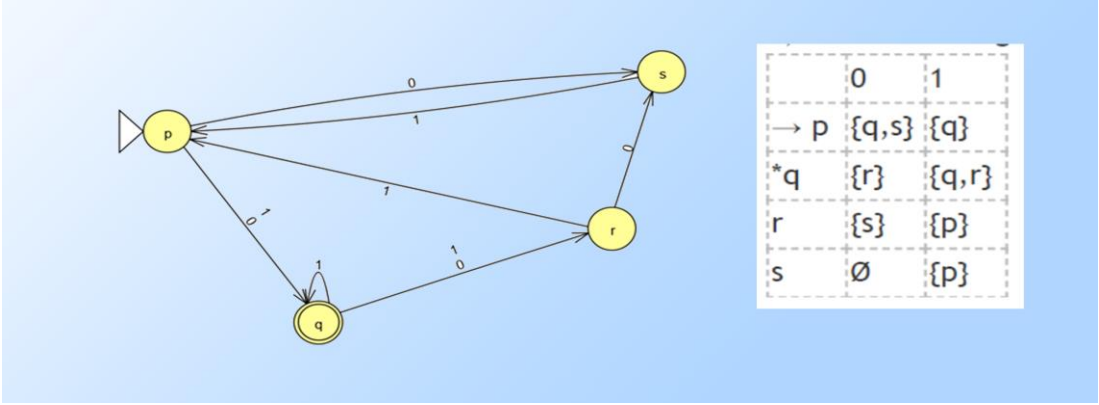


Figura 4 - DFA (aceitação de Strings que não tenham 1 consecutivos)

NFA – Autómatos Finitos Não-Deterministas – semelhantes aos DFA, diferenciando-se na Função, que ao processar uma entrada composta pelo estado corrente e o símbolo lido, tem como resultado um conjunto de novos estados. É representado formalmente por uma 5-tupla (Q, Σ, δ, qo, F), onde:

- Q é um conjunto finito de estados.
- Σ é um conjunto finito de símbolos, Alfabeto.
- δ é a função de transição, isto é, δ: Q x Σ → P(Q).
    - P(Q) conjunto das partes de Q.
- qo é o estado inicial, onde qo ∈ Q.
- F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.

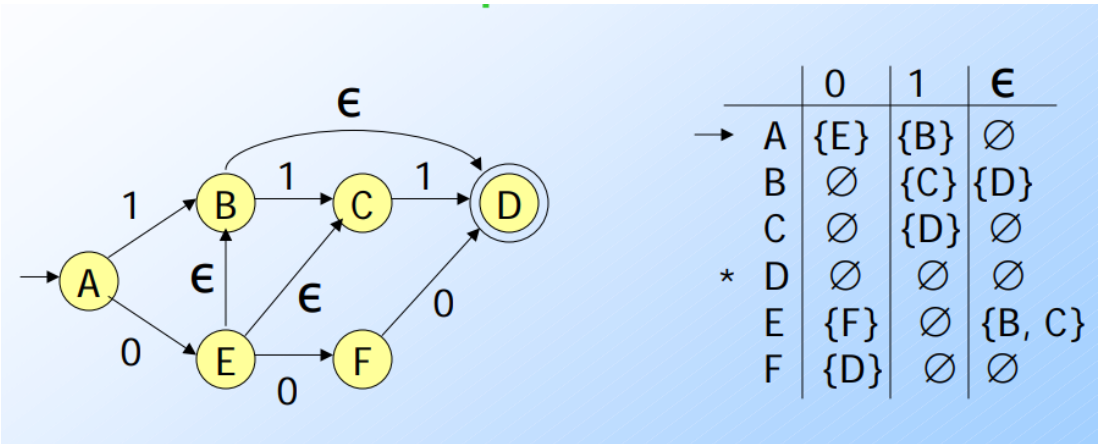


	0	1
→ p	{q,s}	{q}
*q	{r}	{q,r}
r	{s}	{p}
s	Ø	{p}

Figura 5 - NFA e respetiva tabela transições.

NFA-ε – Autómatos Finitos Não-Deterministas com Movimentos Vazios – semelhantes aos NFA, acrescendo a possibilidade de transições de estados sem leitura de um símbolo da Fita. É representado formalmente por uma 5-tupla (Q, Σ, δ, qo, F), onde:

- Q é um conjunto finito de estados.
- Σ é um conjunto finito de símbolos, Alfabeto.
- δ é a função de transição, isto é, δ: Q x Σ → P(Q).
    - P(Q) conjunto das partes de Q.
- qo é o estado inicial, onde qo ∈ Q.
- F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.



		0	1	ϵ
→	A	{E}	{B}	∅
	B	∅	{C}	{D}
	C	∅	{D}	∅
*	D	∅	∅	∅
	E	{F}	∅	{B, C}
	F	{D}	∅	∅

Figura 6 - NFA-ε e respetiva tabela transições.

PDA – Autómato Pilha – também conhecidos como autómatos *push-down* são máquinas semelhantes aos autómatos finitos anteriores, acoplado a uma pilha que funciona como memoria auxiliar, que pode armazenar uma *String* de comprimento arbitrário. A pilha pode ser lida e modificada apenas na parte superior. É representado formalmente por uma 6-tupla (Q, Σ, Γ, Δ, qo, F), onde:

- Q é um conjunto finito de estados.
- Σ é um conjunto finito de símbolos, Alfabeto.
- Γ é um conjunto finito de símbolos, Alfabeto da pilha.
- $\Delta \subseteq (Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*)$ é a relação de transição.
- qo é o estado inicial, onde qo ∈ Q.
- F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.

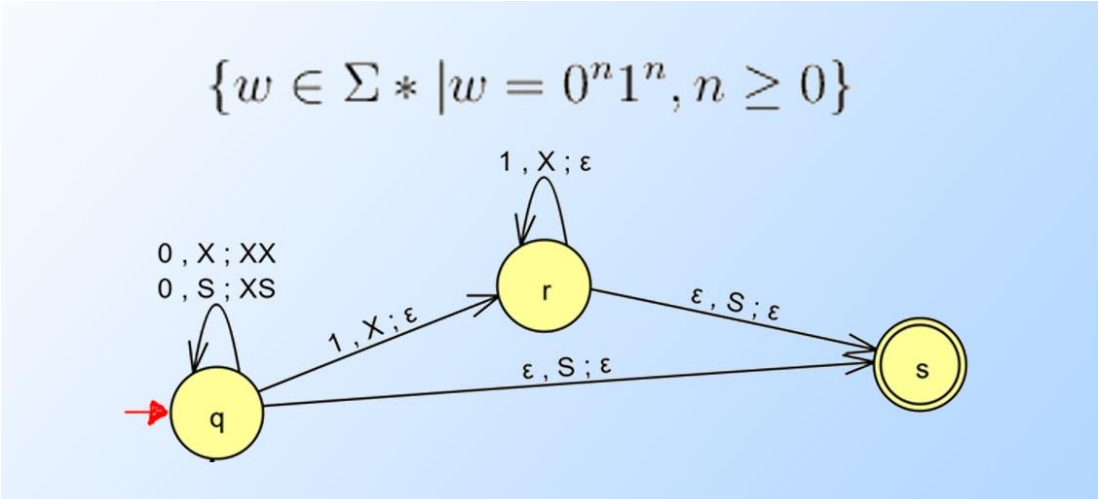


Figura 7 - PDA que reconhece as sequências vazias ou de 0's quando seguida da mesma quantidade de 1's.

TURING – Máquina de Turing – os autómatos desta classe são capazes de executar quaisquer tarefas que sejam efetivamente computáveis. Na MT a Fita é extensível para a esquerda e para a direita, i.e.., a MT possui a Fita necessária à computação onde as células ainda não preenchidas estão preenchidas com um símbolo especial *branco*. A Unidade de Controlo (cabeça) lê e escreve símbolos na fita movendo-se para esquerda ou direita. A Função de Transição além do movimento indica que símbolo deverá ser escrito, quando não existirem entrada na tabela para a combinação atual de símbolo e estado a máquina para. É representada formalmente por uma 7-upla (Q, Σ, Γ, qo, b, F, δ), onde:

- Q é um conjunto finito de estados.
- Σ é um conjunto finito de símbolos, Alfabeto.
- Γ é um conjunto finito de símbolos, Alfabeto da Fita.
- qo é o estado inicial, onde qo ∈ Q.
- b é o símbolo branco, onde b ∈ Γ.

- F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.
- $\delta: Q \times \Gamma \Rightarrow Q \times \Gamma \times \{L, R\}$ é a relação de transição, em que R e L são as direções direita e esquerda, respetivamente.

A figura 8 apresenta um exemplo para o desenho de uma máquina de Turing que toma como entrada um número N e que lhe adiciona 1, onde a fita contém inicialmente um $ seguido por N em binário. A cabeça está inicialmente sobre o $ e no estado q0. A MT para com N + 1 (em binário) na fita, com a cabeça sobre o símbolo mais à esquerda de N+1, no estado q3.

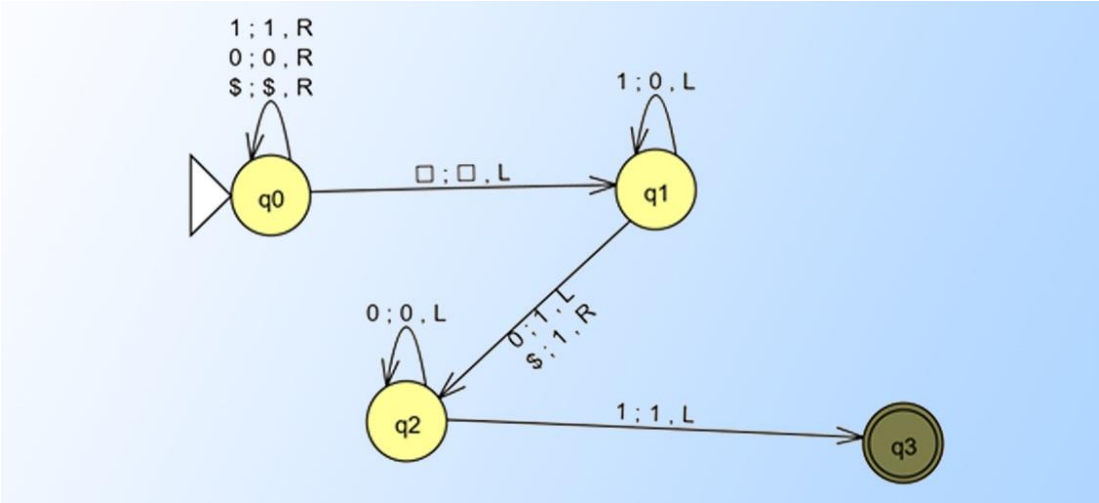


Figura 8 - Exemplo Máquina Turing

A tabela 3 apresenta as transições da MT, no estado q0 move a cabeça para a direita até encontrar símbolo vazio. Estado q1 substitui 1's por 0's, movendo se para a esquerda, até encontrar 0 ou $, mudando para q2 e substituindo o símbolo encontrado por 1. Estado q2 move a cabeça para a esquerda até encontrar primeiro 1, quando passa para estado final.

E/S	0	1	$	B
q_0	(q0,0,R)	(q0,1,R)	(q0,$,R)	(q1,b,L)
q_1	(q2,1,L)	(q1,0,L)	(q2,1,R)	
q_2	(q2,0,L)	(q3,1,L)		
q_3				

Tabela 3 - Exemplo da Tabela de Transições da MT

Representações Estruturais

As linguagens regulares constituem a classe de linguagens com menor poder de representação, sendo possível desenvolver algoritmos de reconhecimento, existindo várias aplicações, como a analise léxica, sistemas de animação, hipertextos e hipermédia (Menezes, 2000). As linguagens regulares podem ser presentadas por um autómato finito e por uma Expressão Regular (REGEX).

REGEX – Expressões Regulares – uma forma sequencial de especificar uma linguagem regular, através de um padrão de *Strings* que descreve o mesmo que pode ser descrito por um autómato finito. Um exemplo, em notação UNIX de uma REGEX "$[A-Z][a-z]*[ ][A-Z][A-Z]$" representa uma palavra iniciada com maiúscula, seguida de espaço e duas maiúsculas, nesta seria aceite a sequência "Porto PT".

GRAMMAR – Gramática Livre de Contexto - são modelos uteis na conceção de software que processa estrutura de dados recursivamente. O exemplo mais conhecido é o "*parsing*", componente de um compilador que lida com os recursos aninhados recursivamente da linguagem de programação, como expressões aritméticas condicionais. É representada formalmente por uma 4-upla (V, T, P, S), onde:

- V é o conjunto de variáveis
- T é o conjunto de terminais
- P é o conjunto de produções
- S o símbolo inicial, que deverá ser elemento de V.

Um exemplo de uma Gramática Livre de Contexto para números binários compostos por n zeros seguidos de n uns é dado por CFG=({S}, [0,1}, {S->01, S->0S1}, {S}).

Existem algumas operações possíveis de realizar sobre autómatos, as quais deverão ser alvo de versões posteriores deste Laboratório, como a conversão NFA-ε para NFA, NFA para DFA, DFA para Expressões Regulares, Expressões Regulares para NFA-ε e Gramática Livre de Contexto para PDA.

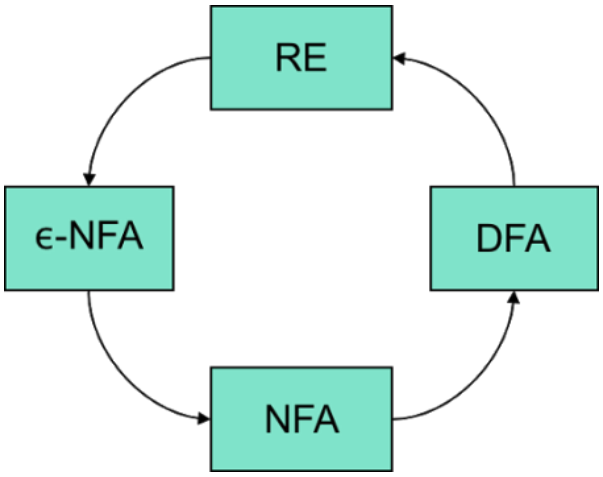


Figura 9 - Ciclo de Conversões

A2.3 – Funcionalidades Globais

De modo a garantir as propostas apresentadas na secção A1.3, o simulador deverá ser implementado em ambiente *web* recorrendo a linguagens de programação simples e universais, para que possa ser garantida a extensibilidade de adaptabilidade, daí as escolhas indicadas na definição de recursos (secção A1.7).

Para a inclusão na plataforma Moodle, existem pelo menos duas hipóteses, a tecnologia *Learning Tools Interoperability* (LTI)[10] e *Inline Frame (*Iframe*)*[11], a primeira amplamente suportada pelo Moodle, enquanto a segundo com limitações. Dado o fato de nesta fase não existir disponibilidade para aprofundamento da primeira, fica a referência para novas versões do UAbALL, optando assim pela solução Iframe, dado que responde às necessidades de implementação e distribuição ao momento, assim como suportada por todos os navegadores *web* utilizados neste projeto, apesar de existirem atributos que não são suportados com HTML5, como os referentes a alinhamento, barras *scroll* e margens.

A2.4 – Registo Regras DFA

Tomando em conta a definição formal apresentada secção A2.2.1, este modulo deverá permitir construir um DFA, através da recolha dos seguintes dados:

- Conjunto de Estados.
- Símbolos que compõe o alfabeto
- Função de Transição
- Identificação do Estado Inicial e Final

Com a construção do DFA deverá ser possível a simulação de sequências com visualização passo-a-passo, ou mesmo através de um método mais rápido para quem observa.

Outro aspeto a considerar na Implementação é a possibilidade de construir autómatos a partir da leitura de ficheiros, que respeitem um *template* pré-definido e simular os mesmos, mas também a possibilidade de salvar para ficheiro um DFA construído diretamente na plataforma.

A2.5 – Componentes Interface Gráfica

A interface tem como função permitir ao utilizador obter uma visão panorâmica do conteúdo, navegando neste sem perda de orientação, navegando de acordo com o seu interesse. As suas funções são a condução, orientação, e responder às interações.

Para a condução e orientação deverão ser utilizados *containers* através de etiquetas (*tags*) <*div*> HTML para a divisão de conteúdos estilizados através de CSS, permitindo assim uma abstração na construção dos conteúdos, agilizando simultaneamente os processos de navegação, enquanto o CSS tratará da capacidade *Responsive* da plataforma.

Para as interações tomaremos tarefas implementadas com recurso ao JavaScript, para recolha e processamento de dados.

[10] https://www.imsglobal.org/basic-overview-how-lti-works
[11] https://docs.moodle.org/38/en/Iframe

Na simulação do autómato a biblioteca Graphviz para SVG, permitirá implementar o grafo interativo, visualizando o DFA passo-a-passo. A figura 10 Figura 10 - Simulação DFA apresenta uma simulação da interação para uma String que contem dois uns consecutivos.

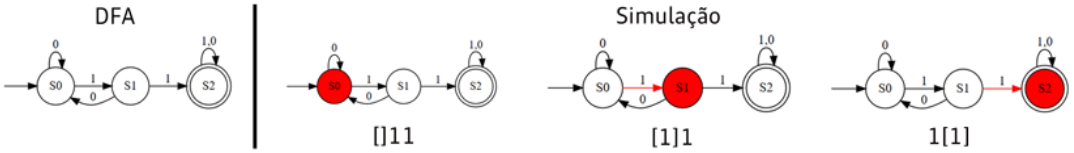


Figura 10 - Simulação DFA (aceitação de Strings que contêm dois uns consecutivos)

Capítulo 3

B- Implementação

Concluído o planeamento passamos à construção e codificação da plataforma. Neste capítulo discutimos a implementação das especificações desenhadas no capítulo anterior, para o cumprimento dos objetivos delineados.

B1. DESENVOLVIMENTO

Nas secções seguintes são apresentadas as principais decisões tomadas para cada um dos temas levantados durante a fase de conceção. Iniciaremos com a implementação dos idiomas, seguindo posteriormente para os menus, na qual foi tomada uma primeira decisão para a rentabilização da produção e futuras necessidades de manutenção e expansão com a implementação dos blocos contendo o resumo científico para cada um dos temas. Nas funcionalidades globais é apresentada uma decisão de incremento aos recursos, com a utilização da *framework* THREE.js para aplicar um *container* com uma animação para os blocos que ainda não contêm simulador disponível. A secção para as funcionalidades DFA apresenta a implementação do simulador disponibilizado nesta primeira versão destacando os aspetos mais particulares da operacionalidade. Por fim, é apresentada a implementação das componentes gráficas, particularizando as decisões tomadas para o CSS.

B1.1 – Idiomas

```
<script type="text/javascript">
    function idioma() {
        switch (navigator.language.substring(0, 2)) {
            case "en":
                window.location.href = "./start.html?lang=EN";
                break;
            case "es":
                window.location.href = "./start.html?lang=ES";
                break;
            default:
                window.location.href = "./start.html?lang=PT";
                break;
        }
    }
</script>
```

Figura 11 - Script da função idioma

A implementação do idioma de abertura foi produzida com recurso ao método *substring*() de JavaScript para *String* para a recolha do código de idioma (Tabela 2), e aplicando o comando *switch case* em conformidade com o fluxo apresentado na figura 2.

Foi acrescentada uma variável de idioma ao URL, para agilizar o processamento desta na aplicação (https://andremaciel.pt/UAb/UAbALL/start.html**?lang=PT**). Esta variável será utilizada por toda a aplicação na escolha do idioma a ser apresentado no ecrã. Esta funcionalidade também permite a economia de recursos, uma vez que passa ser possível a construção multilingue com recurso a esta funcionalidade, eliminando a necessidade de construção de páginas separadas para cada idioma.

Os processos de ativação, manutenção e utilização da variável referente ao idioma, são apresentados de seguida:

- Ativação – Realizada aquando da entrada no site e na opção da seleção de idioma do menu, através da variável "lang" no URL, utilizando para tal a função GET e passando os parâmetros por um REGEX para o efeito, conforme a figura *12*.

```
function $_GET(param) {
    var vars = {};
    window.location.href.replace(location.hash, "").replace(
        /[?]+([^=&]+)=?([^&]*)?/gi, // regexp
        function (m, key, value) {
            // callback
            vars[key] = value !== undefined ? value : "";
        }
    );
    if (param) {
        return vars[param] ? vars[param] : null;
    }
    return vars;
}
```

Figura 12 - Função para obtenção parâmetro" lang"

- Manutenção – Esta é realizada através do armazenamento numa variável global do JavaScript (**var** lang), para assim poder ser utilizada na navegação sem necessidade de estar constantemente a ler o URL.

- Utilização – O idioma é mantido atualizado através da utilização da variável global armazenada que é concatenada ao endereço de destino, que por sua vez carregará os componentes com a identificação correspondente ao parâmetro *lang*. As figuras *13 e 14* exemplificam esta abordagem.

```
<p id="PT">Projecto de Engenharia Informática 2020 de … </p>
<p id="EN">Informatics Engineering Project 2020 of … </p>
<p id="ES">Proyecto de Ingeniería Informática 2020 de …</p>
```

Figura 13 - Exemplo utilização parâmetro lang

```
function main(body) {
    body = body || home; //se nulo atribui home

    $(document).ready(function () {
        if (!lang) {
            lang = $_GET("lang");
        }
        $("#nav").load(menu);
        $("#corpo").load(body + lang);
        $("#footer").load(footer + lang);
    });
}
```

Figura 14 - Função para carregamento de conteúdos em conformidade com idioma

Os anexos [1] a [3] apresentam o detalhe da codificação desta implementação multilingue.

B1.2 – Menus

Contrariamente ao que havia sido discutido no capítulo anterior, a descrição científica de cada uma das opções disponíveis nos menus não será colocada na página de entrada. Para a página Home ficou reservada a apresentação do logotipo do projeto, assim como para os logos das tecnologias utilizadas no mesmo.

Cada opção do menu carregará para o corpo da aplicação a respetiva descrição científica do autómato, gramática ou máquina de *Turing*, reservando o *container* para a aplicação gráfica da construção do simulador. Esta opção é tomada devido ao caracter educacional da aplicação, sendo considerado que a base teórica é importante para a melhor compreensão do processo de construção prático, pelo que com este processo é garantida a presença continua da descrição teórica.

Os menus são implementados com recurso ao carregamento por JavaScript dos conteúdos em conformidade com o idioma de navegação (discutido na secção anterior), neste caso para o carregamento da *div* **corpo**. Para este processo são guardadas como variáveis globais, no ficheiro *loader.js*, os endereções dos blocos correspondentes a cada opção do menu, como poderá ser observado na figura *15*.

Cada variável inclui o respetivo endereço do conteúdo, assim como o marcador para o identificador do idioma a carregar, para que deste modo possamos ter o conteúdo num único bloco de informação, a cada respetiva funcionalidade, otimizando, deste modo, a

adaptabilidade e expansibilidade multilingue de cada opção implementada e que no futuro necessite de ser intervencionada.

Esta capacidade de carregamento de conteúdos é conseguida com a implementação do *jQuery*, mais propriamente com o recurso ao método *load*() que carrega dados de um local do servidor e coloca os dados devolvidos no elemento selecionado (figura 14).

```
var body; //corpo da página
var menu = "./blocos/menu.txt";
var footer = "./blocos/footer.txt #";
var home = "./blocos/corpo.txt #";
var dfa = "./blocos/dfa.txt #";
var nfa = "./blocos/nfa.txt #";
var enfa = "./blocos/enfa.txt #";
var pda = "./blocos/pda.txt #";
var turing = "./blocos/turing.txt #";
var regex = "./blocos/regex.txt #";
var grammar = "./blocos/grammar.txt #";
```

Figura 15 - Definição dos endereços das opções

B1.3 – Funcionalidades Globais

Para cumprimento da pretensão do UAbALL ser um laboratório virtual integrado de simulação das linguagens formais, este foi implementado na plataforma moodle, e de igual modo adaptável a qualquer plataforma de ensino digital, utilizando *Inline Frame* [figura 16], resultando na apresentação gráfica da figura 17.

```
<iframe height="600" width="100%" src="https://andremaciel.pt/UAb/UAbALL">
  <p>Your browser does not support iframes.</p>
</iframe>
```

Figura 16 - Inline Frame

Apesar de não ter estado previsto nas especificações iniciais foi adicionado um bloco gráfico, para os simuladores não disponibilizados nesta versão, de "*Em Construção…*", esta decisão foi tomada não só pelo caracter estético para a aplicação, mas pelo facto de a framework **Three.js** utilizada, ser recente na unidade de Computação Gráfica da UAb, sendo deste modo mais um meio de promoção do seu estudo.

B1.3.1 – Three.js

Three.js é uma framework em JavaScript, desenvolvida a partir do *WebGL* (versão Web da biblioteca *OpenGl*) que permite criar e renderizar cenas 3D diretamente no navegador.

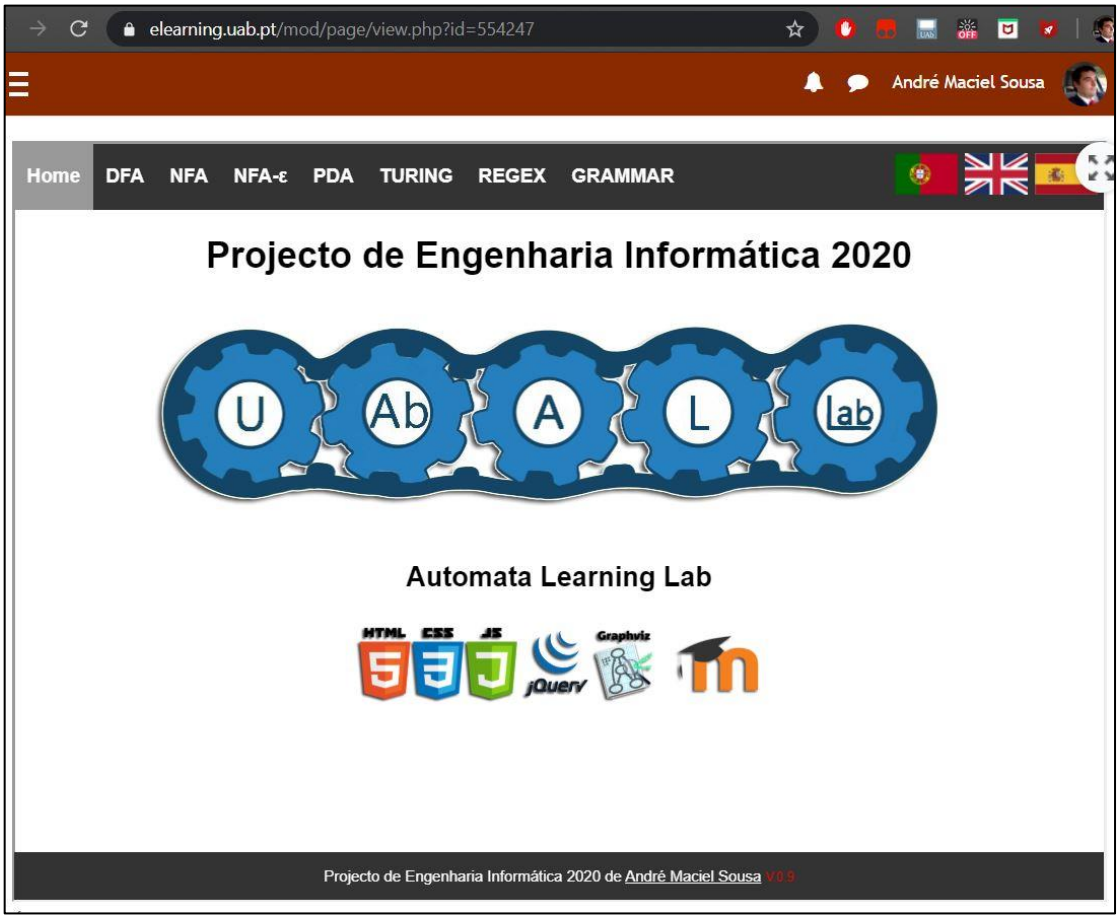


Figura 17 - UABALL no Moodle UAb

Toda a aplicação Three.js tem os seguintes componentes básicos:

Scene – O suporte para todo o "universo" que será criado com os objetos 3D. A cena permite a configuração do ambiente em que os objetos vão ser renderizados. É neste cenário que são inseridos os objetos, as luzes e as câmaras da cena

Camera – Equivalente ao mundo real, é utilizada para visualizar a Scene. O Three.js possui três tipos de câmaras: CubeCamera, PerspectiveCamera e OrthographicCamera, sendo as duas últimas as mais utilizadas, e das quais definimos de seguida.

A diferença básica está na noção de profundidade, uma vez que na PerspectiveCamera a medida que os objetos se afastam em direção ao fundo da cena, eles vão modificando de tamanho, muito mais próximo da forma como vemos o mundo real. Enquanto que na OrthographicCamera temos uma ideia de fundo fixo, onde objetos distantes na cena apresentam a mesma proporção de objetos do mesmo tamanho que estejam próximos na cena, este tipo de câmara é muito utilizada em cenários 2D (figura 18).

Conforme o tipo de câmara que estejamos a utilizar os parâmetros serão diferentes.

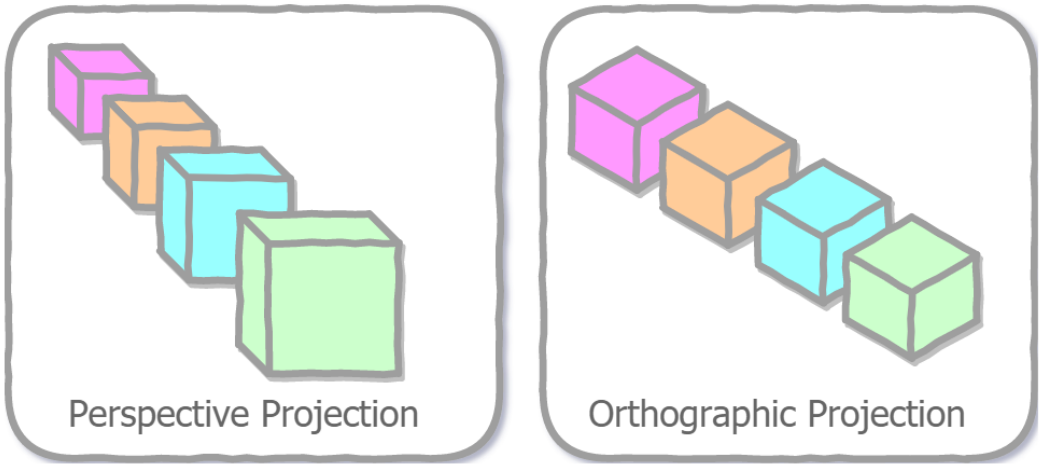


Figura 18 - Tipos de Projeção

PerspectiveCamera*(fov, aspect, near, far):*

fov: Este parâmetro representa o vertical field of view (campo de visão vertical). Ele define o que pode ser visto pela câmara. Nós humanos temos um capo de visão de aproximadamente 180 graus. O valor é passado em graus num intervalo de 1 a 179.

aspect: Este parâmetro representa o *aspect ratio* da tela. Ele é calculado dividindo a largura da tela pela sua altura.

near: Este parâmetro define a menor distância da câmara permitida para renderização dos objetos. Normalmente, esse valor é muito pequeno não podendo ser zero. Isso permite que objetos que estejam muito próximos da câmara apareçam na cena.

far: Este parâmetro define a maior distância da câmara permitida para renderização dos objetos. Normalmente, esse valor varia de 500 a 2000. Isso permite que objetos que estejam muito distantes da câmara não sejam renderizados. Valores muito altos podem prejudicar a performance do sistema.

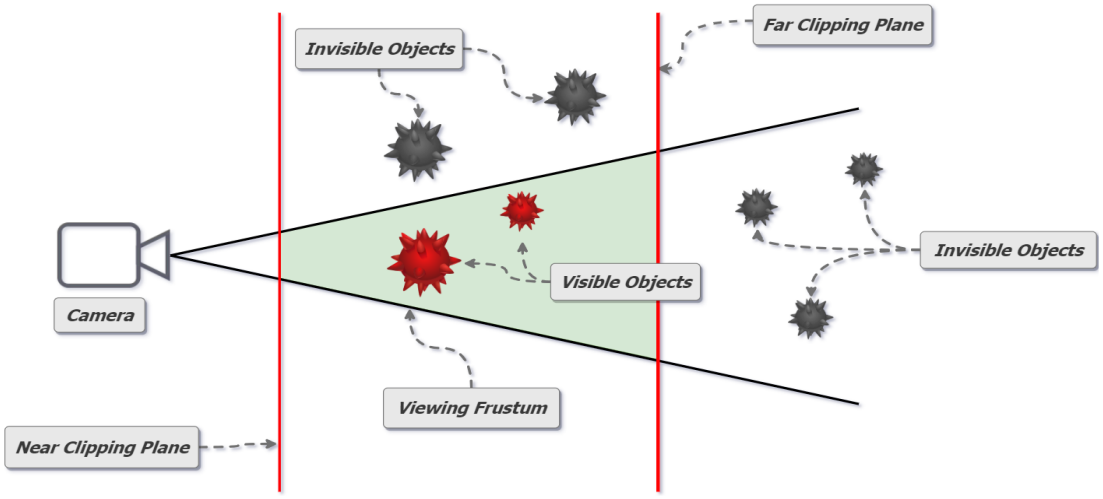


Figura 19 - Tipos de Projeção

OrthographicCamera *(left, right, top, bottom, near, far)*

left, right, top, bottom: determinam o limite da tela que será renderizado, por exemplo, se definir o left -100. Qualquer objeto que esteja mais a esquerda do que esse limite não será renderizado. A mesma ideia se aplica aos demais parâmetros.

near: Este parâmetro define a menor distância da câmara permitida para renderização dos objetos. Normalmente, esse valor é muito pequeno. Isso permite que objetos que estejam muito próximos da câmara apareçam na cena.

far: Este parâmetro define a maior distância da câmara permitida para renderização dos objetos. Normalmente, esse valor varia de 500 a 2000. Isso permite que objetos que estejam muito distantes da câmara não sejam renderizados. Valores muito altos podem prejudicar a performance do sistema.

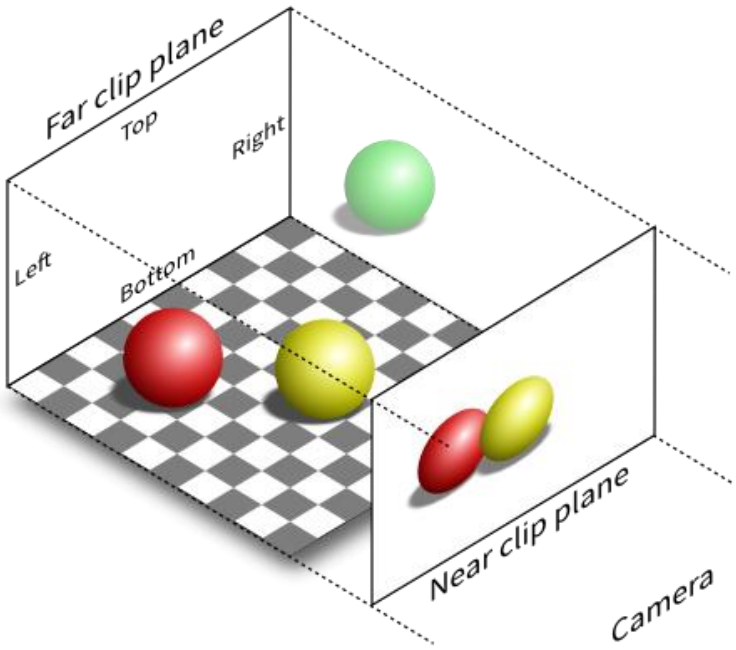


Figura 20 - OrthographicCamera

Canvas – Este é um elemento de tela HTML e, assim como uma tela no mundo real começa com retângulo branco.

Renderer – Esta é uma máquina que utiliza uma câmara e uma cena como entrada e renderiza para a tela o "universo" criado. Juntos, a Scene, a Camera, a Canvas e o Renderer fornecem a estrutura de um aplicativo, no entanto, nenhum deles pode realmente ser visto.

Mesh – Adicionável à cena e define a posição no espaço 3D, é um suporte para uma geometria e um material.

Geometry – Define a geometria da Mesh.

Material – Define a aparência da superfície da Mesh.

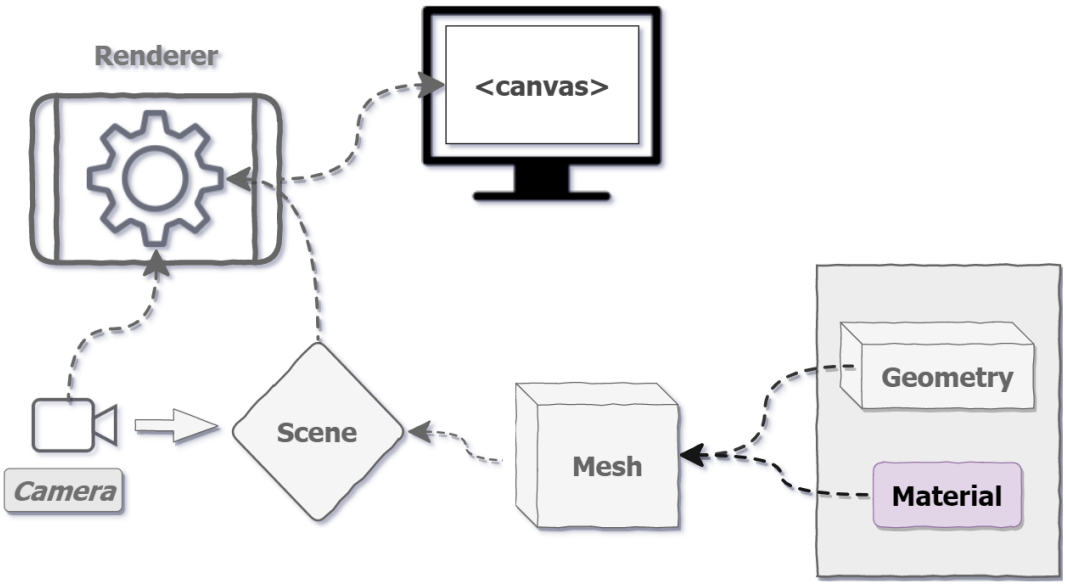


Figura 21 - Componentes de um aplicativo 3D em tempo real

Implementação THREE.JS

O arquivo *three.js* chamado no *start.html* (anexo[3]) contém toda a API para criação e manipulação dos objetos 3D, para este projeto foi criado o ficheiro *underC.js* (anexo [6]) referenciado no mesmo arquivo *html*.

O arquivo *underC.js* contem todas as informações para a criação dos objetos, precisamos de quatro elementos: uma cena (*scene*), uma câmara (*camera*), um renderizador (renderer) e um contentor (container).

```
var scene; //representa um container onde vai ser colocado todos os objetos que queremos renderizar.
var camera; //representa o campo de visão da cena.
var renderer; //será nosso renderizador dos objetos contidos na cena.
var container; //referencia ao elemento que apresentara a scene
```

Figura 22 - Elementos básicos Three.JS

O primeiro passo foi a criação das variáveis globais para cada um dos elementos que precisamos de representar. Instanciados os elementos necessários, aplicamos a renderização à cena criada através da função *animate* que é chamada no arquivo *loader.js* através da função *underConstr* (esta função integra a capacidade de adicionamento de nova div nos blocos específicos, aqueles que ainda não estão construídos, explicados mais à frente – Adicionar Em Construção).

A chamada para a função *init()* da aplicação Three.js, inclui a variável *lang* ativa, para que possa ser utilizada na disponibilização multilingue da mensagem "Em construção... próximas versões."

A função init() cria a referencia ao elemento que apresentará a cena, um objeto scene, que será posteriormente adicionado ao render, a câmara virtual com projeção em perspetiva, chama as funções para criação dos objetos, adiciona a luz ambiente ao objeto scene e inicia o renderizador.

```
function init(lang) {
    container = document.querySelector("#mycanvas");
    scene = new THREE.Scene();
    // Criação de camera virtual com projeção em perspectiva, que vai ser depois
adicionada ao objeto render
    camera = new THREE.PerspectiveCamera(
        45,
        container.clientWidth / container.clientHeight,
        0.1,
        1000
    );
    camera.position.set(0, 0, 60);
    camera.lookAt(0, 0, 0);
    // criacao dos objectos
    constroiObj();

    // coloca texto, em posição centrada e minima para versão telemovel
    // em conformidade com o idioma de navegação

    if (lang == "EN") {
        escreveTxt("Under construction...", "next versions.");
    } else if (lang == "ES") {
        escreveTxt("En construcción...", "próximas versiones.");
    } else {
        escreveTxt("Em construção...", "próximas versões.");
    }
    // Luz Ambiente
    luzAmbiente = new THREE.AmbientLight(0xffff00, 1);
    scene.add(luzAmbiente);

    // Renderizador
    renderer = new THREE.WebGLRenderer({ antialias: true, alpha: true });
    renderer.setSize(container.clientWidth, container.clientHeight);
    //definir proporção correta de pixels para o dispositivo em que está sendo
executado
    renderer.setPixelRatio(window.devicePixelRatio);
    container.append(renderer.domElement);
}
```

Figura 23 - Função init() Three.js

Na criação dos objetos tomando partido de funcionalidades básicas da biblioteca Three.js, como método *Group* que permite a agregação de um conjunto de objetos, podendo deste modo aplicar transformações ao grupo como se fosse um único objeto. Como objeto foi criado um átomo utilizando a geometria da esfera e uma textura como material, posteriormente através de um ciclo foram criados diversos clones, em diferentes posições numa tentativa de passar a ideia de uma "malha" de átomos.

```
function constroiObj() {
    atom = new THREE.Group();

    var geometry = new THREE.SphereBufferGeometry(10, 32, 32);
    var material = new THREE.MeshPhongMaterial({
        map: new THREE.TextureLoader().load("imgs/atom.jpg"),
        side: THREE.DoubleSide,
    });
    var aux = new THREE.Mesh(geometry, material);
    atom.add(aux);

    // Alguns clones
    for (i = 30; i < 90; i += 30) {
        var aux2 = aux.clone();
        aux2.position.set(-i, 0, -i / 5);
        atom.add(aux2);
        var aux3 = aux.clone();
        aux3.position.set(i, 0, -i / 5);
        atom.add(aux3);
    }

    aux = atom.clone();
    aux.rotation.z = Math.PI / 2;
    atom.add(aux);
    aux = atom.clone();
    aux.rotation.z = Math.PI / 4;
    atom.add(aux);

    scene.add(atom);
}
```

Figura 24 - Função constroiObj() Three.js

A função animate() aplica rotação, em radianos, ao grupo de objetos *atom* a cada um dos eixos x, y e z de 0.002 rad (0.1146 °), a cada chamada ciclo da recursividade. Isto cria um *loop* que faz com o renderizador desenhe a cena sempre que a tela é atualizada, o que tipicamente ocorre 60 vezes a cada segundo. O *requestAnimationFrame* acrescenta a vantagem de pausa do navegador sempre que o utilizador navega para outra aba do navegador, não desperdiçando poder de processamento.

```
function animate() {
    atom.rotation.x += 0.002;
    atom.rotation.z += 0.002;
    atom.rotation.y += 0.002;
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
```

Figura 25 - Função animate() Three.js

Adicionar Em Construção

Para os blocos carregados no corpo da aplicação que ainda não tenham o simulador construído, o *loader.js* incluí uma funcionalidade para a ativação de uma nova *div,* esta funcionalidade começa por eliminar as *div* com a mesma referência que tenham sido ativadas anteriormente para evitar replicação e repetição de conteúdo, e posteriormente chamam a função de criação da nova. A chamada leva a variável global do idioma que será utilizada na aplicação para a opção de idioma ativa no momento da sua chamada, após a criação da *div,* são chamadas as funções *init* e *animate* do arquivo *underC.js* responsáveis pela construção da animação 3D apresentada anteriormente.

```
...
// adicionar nova div para animacao "proximas versoes"
        $("#mycanvas").remove();
        if (body != home && body != dfa) {
            cria_newDiv(lang);
...

// adicionar nova div
function cria_newDiv(lang) {
    var novadiv = document.createElement("div");
    novadiv.setAttribute("id", "mycanvas");
    document.body.append(novadiv);
    underConstr(lang);
}

// Em construção animação Three.JS
function underConstr(lang) {
    init(lang);
    if (!startedAnimation) {
        startedAnimation = true;
        animate();
    }
}
```

Figura 26 - Ativação div e Three.js Em Construção

Esta implementação serviu de ensaio para alguns aspetos, como o caso da ativação do *container* via JavaScript e a utilização de componentes específicos de cada idioma disponibilizado, que serão necessários incluir na construção do simulador DFA, que será discutido na secção seguinte.

B1.4 – Funcionalidades DFA

O modulo DFA permite contruir e validar um autómato desta natureza, mostrando que estamos perante um DFA. Na secção A2.21 foi apresentada a definição formal de DFA.

A figura 27 propõe um mapa como guia da apresentação das funcionalidades implementadas através do desenvolvimento disponibilizado nos anexos *dfa.txt* e *dfa.js*.

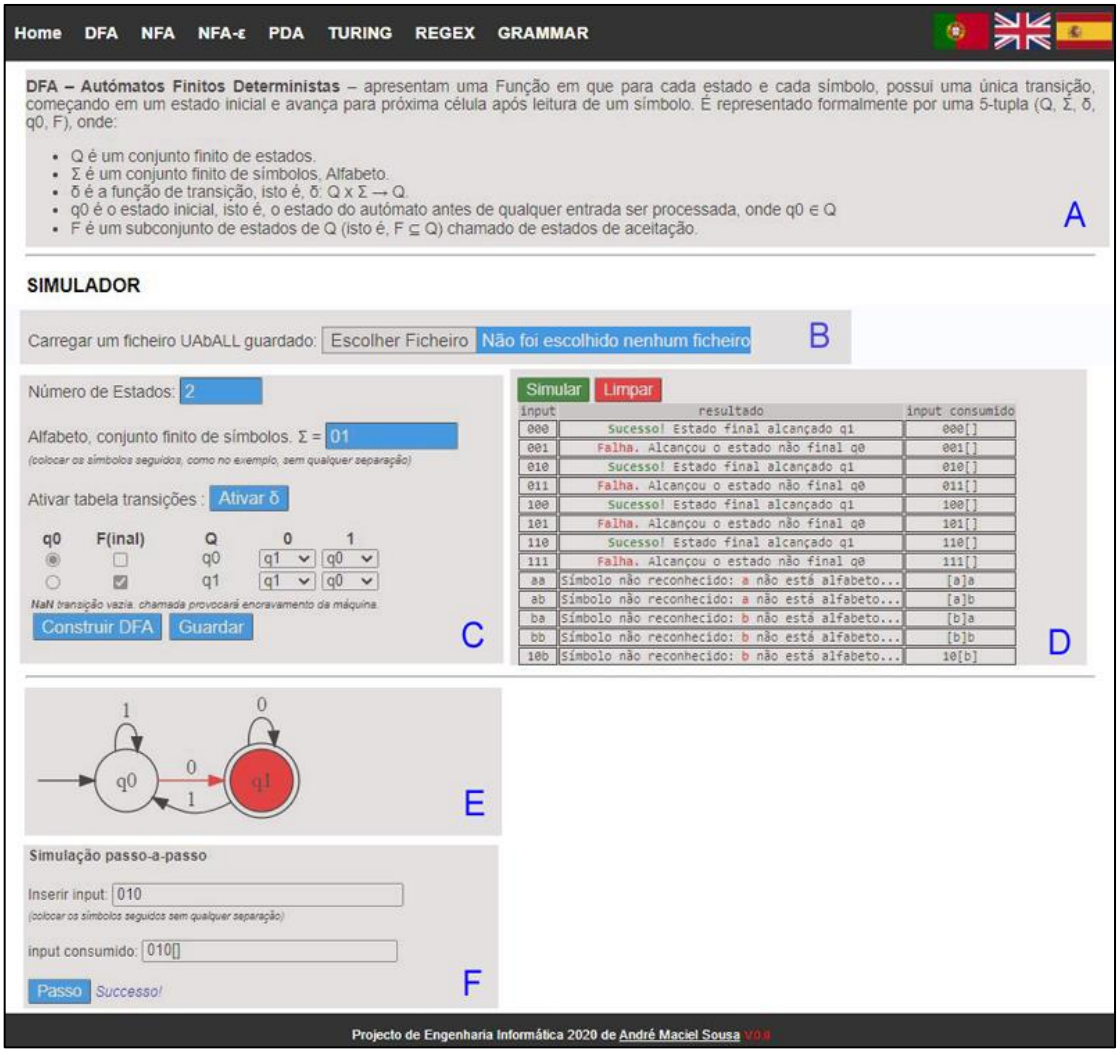


Figura 27 - Mapa Apresentação DFA

Toda a construção é iniciada com o bloco *dfa.txt* onde são definidas as áreas apresentadas na figura 27. Este processo é iniciado aquando da chamada em menu da opção DFA, nos parágrafos seguintes descreveremos cada uma destas zonas.

O bloco *dfa.txt* começa pelo carregamento da descrição teórica do autómato (zona A), e disponibilizando os conteúdos iniciais para a construção do autómato. A primeira opção é a possibilidade de carregamento de um ficheiro gerado anteriormente no laboratório[12] (zona B), esta opção é ocultada quando iniciada a construção do autómato no passo seguinte, a descrição técnica será realizada na secção **Carregamento de Ficheiro**.

Na zona C temos a recolha dos dados que completam o quinteto que define o DFA: número de estados, conjunto finito de símbolos que compõe o alfabeto, estado inicial q0, estados de aceitação ou finais e função de transição; terminando com a opção de guardar o autómato detalhado e com ativação da construção do DFA. O detalhe técnico das funcionalidades envolvidas nesta zona será discutido nas secções **Contruir Transições**, **Gravar Ficheiro** e **Criação de Gráfico**.

Na zona D é fornecida a possibilidade de testar diferentes palavras com o autómato de um modo rápido e com apresentação do resultado, assim como qual o último input consumido, para que o utilizador possa verificar em que momento ocorreu a paragem. O processo técnico é descrito na secção **Tabela de Simulações**.

Na Zona E será apresentado gráfico criado, que inclui uma interação com a simulação passo-a-passo disponibilizada na zona F e cuja funcionalidade será apresentada na secção **Passo-a-Passo**.

B1.4.1 – Estrutura do ficheiro UAbALL **Descrição dos campos**	**Figura 4**	**Figura 10**
Número de Estados	3	3
Alfabeto	01	01
Estado Inicial	0	0
Estados Finais: *1 estado Final, 0- não final*	110	001
Tabela de transições: *colunas – alfabeto, linhas – estados*	0;1	0;1
	0;2	0;2
	2;2	2;2
Sequências a simular (1 por linha)	000	000
	001	001
	010	010
	011	011
	101	101
	aa	aa
	ab	ab
	ba	ba
	bb	bb

Tabela 4 - Composição ficheiro UAbALL

[12] Sendo o ficheiro em modo texto, este pode também ser criado ou modificado manualmente.

A tabela 4 compara a constituição do ficheiro UAbLL dos autómatos das *figuras 4 e 10* que têm como objetivos o reconhecimento de sequências que não tenham 1's consecutivos, ou tenham 1's consecutivos, respetivamente.

A primeira linha contém o número de estados do autómato, a segunda o alfabeto, a terceira o estado inicial, a quarta quais os estados finais(em formato binário), a quinta e seguintes (a quantidade depende do numero de estados) a tabela de transições e por fim as sequências a simular (uma linha em branco corresponderá à sequência vazia).

B1.4.1 – Carregamento de Ficheiro

O carregamento de um ficheiro previamente guardado no UAbALL é processado com a chamada da função *ler_UAbALL()* que inicia com o carregamento, como texto para processamento com recurso ao objeto *FileReader*[13], primeiro passo da função apresentada na figura 29 Figura 29 - Função ler_UAbALL()com a chamada da função da figura 28. Começamos por obter a informação do ficheiro a carregar e um manipulador para o evento *load* que é chamado cada vez que a operação de leitura é completada com sucesso, enquanto que o método *readAsText* inicia a leitura do conteúdo e quando terminado temos como resultado o conteúdo do arquivo como *string* de texto.

```
function loadFileAsText() {
    var fileToLoad = document.getElementById("fileToLoad").files[0];

    var fileReader = new FileReader();
    fileReader.onload = function (fileLoadedEvent) {
        buffer = fileLoadedEvent.target.result;
    };
    fileReader.readAsText(fileToLoad, "UTF-8");
}
```

Figura 28 - Função loadFileAsText()

A função *ler_UAbALL()* passa então a processar esta string de texto, linha a linha, onde a primeira contem o numero de estados, a segunda o alfabeto, na terceira o indexante do estado inicial que passamos como inteiro. Os estados de aceitação são lidos na quarta linham sendo que estão marcados a 1 se finais e pela ordem crescente do conjunto de estados. Da quinta linha até à linha indexada pelo número de estados é verificada a tabela de transições para cada estado, que estão separados por ";", para a sequência e símbolos pela ordem que foram carregados. Por fim, e para cada uma das restantes linhas, guardamos cada linha como uma palavra que será a nossa base de testes a utilizar na tabela de simulação do nosso DFA, a zona D referenciada na secção anterior.

[13] https://developer.mozilla.org/pt-BR/docs/Web/API/FileReader

```
function ler_UAbALL() {
    loadFileAsText();
    if (lang == "EN") { alert("File loaded.");
    } else if (lang == "ES") {  alert("Archivo cargado.");
    } else {alert("Ficheiro carregado.");
    }
    linhas = buffer.split("\n");
    nEstados = parseInt(linhas[0]);
    document.getElementById("nEstados").value = linhas[0];
    document.getElementById("symbols").value = linhas[1];
    estadoInicial = parseInt(linhas[2]);
    constroi_transitions();
    for (var i = 0; i < nEstados; i++) {
        if (linhas[3].charAt(i) == "1") {
            document.getElementById("Final" + i).checked = true;
        }
    }
    for (i = 0; i < nEstados; i++) {
        colunas = linhas[4 + i].split(";");
        for (j = 0; j < linhas[1].length; j++) {
            if (colunas[j] != "") {
                var val = document.getElementById("X_" + i + "," + j);
                val.selectedIndex = parseInt(colunas[j]) + 1;
            }
        }
    }
    var resto = "";
    for (var i = nEstados + 4; i < linhas.length; i++) {
        resto = resto + "\n" + linhas[i];
    }
    base_tests = resto;
    inOutDiv();
}
```

Figura 29 - Função ***ler_UAbALL()***

B1.4.2 – Gravar Ficheiro

Para o download de um ficheiro com os dados necessários para uma reutilização do projeto de autómato criado anteriormente, é disponibilizada a funcionalidade de gravação de autómato.

Na *Figura 30* encontramos o detalhe do código desta função, que começa por ler o número de estados, seguido dos símbolos que compõe o alfabeto. Iniciada a string para

```
function grava_UAbALL() {
    nEstados = parseInt(document.getElementById("nEstados").value);
    alfabeto = document.getElementById("symbols").value;
    UAbALL_text = "";
    UAbALL_text =
        UAbALL_text + nEstados + "\n" + alfabeto + "\n" + estadoInicial + "\n";
    for (i = 0; i < nEstados; i++) {
        if (document.getElementById("Final" + i).checked) {
            UAbALL_text = UAbALL_text + "1";
        } else {
            UAbALL_text = UAbALL_text + "0";
        }
    }
    UAbALL_text = UAbALL_text + "\n";
    for (i = 0; i < nEstados; i++) {
        for (j = 0; j < alfabeto.length; j++) {
            var val = document.getElementById("X_" + i + "," + j);
            var strUser = val.options[val.selectedIndex].value;
            to_state = "";
            if (strUser != "") {
                to_state = parseInt(strUser.substring(1));
            }
            if (j > 0) UAbALL_text = UAbALL_text + ";";
            UAbALL_text = UAbALL_text + to_state;
        }
        UAbALL_text = UAbALL_text + "\n";
    }
    UAbALL_text = UAbALL_text + base_tests;
...
    if (fich != null) {
        if (fich == "") fich = "UAbALL.txt";
        // texto a salvar, nome do ficheiro
        saveTextAsFile(UAbALL_text, fich);
    }
}
```

Figura 30 - Função grava_UAbALL()

guardar os dados são adicionados a esta, e pela seguinte ordem, sempre seguido de uma quebra de linha ("\n"), o número de estados, o conjunto de símbolos que compõe o alfabeto, o indexante do estado inicial, identificação dos estados finais marcando estes com um 1 e os restantes com 0, para cada estado é guardado o conjunto de transições

por cada símbolo, por fim é guardada a informação constante na base de testes da tabela de simulação.

Por fim o ficheiro de texto é descarregado com recurso ao objeto *Blob*[14] e à função *createObjectURL()*[15], o construtor *Blob* retorna um objeto *Blob* cujo conteúdo consiste na concatenação do *array* de valores passados por parâmetro, neste caso contendo a string *UAbALL_text*. De seguida é criado um objeto *textToSaveAsURL* com a função *createObjectURL()* com o texto a salvar.

B1.4.3 – Tabela de Simulações

Quando ativamos a construção da tabela de transições é realizada uma chamada para a função detalhada na *figura 36,* que disponibiliza a tabela de simulações. Esta tabela contém uma área para acrescentar palavras a testar, assim como um botão para correr simulação e outro para limpar a simulação, permitindo reiniciar todo este processo. Quando ativada a opção limpar é realizada uma nova chamada à função *inOutDiv()* realizando uma reposição para o estado original da tabela e desativando o botão limpar. Esta desativação de botões permite um maior controlo sobre funcionamento sendo produzido no botão de simulação pelo mesmo princípio, isto é, depois de ativado o botão simulação não é mais possível utilizar enquanto não for pressionado o de limpar.

```
function limpar() {
    document.getElementById("run_simul").disabled = false;
    //-- Tratamento de Outputs Ativa/esconde Container--
    inOutDiv();
    document.getElementById("limp_simul").disabled = true;
}
```

Figura 31 - Função limpar ()

A simulação é realizada palavra a palavra, passando cada uma dessas pela função *simular_1()*. Cada palavra é corrida caracter a caracter pela máquina de estados, enquanto não for encontrada uma clausula de paragem, como por exemplo um caracter que não tem correspondência no alfabeto, ou uma transição inexistente para o símbolo que esta em leitura. Quando chegados na última posição da palavra, caso estejamos num estado final (de aceitação) é chamada a função de produção de mensagens multilingue, *resultMG()* (disponível no anexo *dfa.js*), com informação de sucesso(1), caso contrario com informação de insucesso (0).

[14] https://developer.mozilla.org/pt-BR/docs/Web/API/Blob

[15] https://developer.mozilla.org/pt-BR/docs/Web/API/URL/createObjectURl

```
function simular_1(para_input) {
    var alfabeto = document.getElementById("symbols").value;
    var nEstados = parseInt(document.getElementById("nEstados").value);
    var stopit = 0;
    var primeiro = 0;
    var prev_stat = -1;
    var curr_stat = -1;
    var pos = -1;
    while (stopit == 0) {
        if (prev_stat == -1 && curr_stat == -1) {
            curr_stat = estadoInicial;
            primeiro = 1;
            continue;
        }
        pos++;
        if (pos == para_input.length) {
            if (document.getElementById("Final" + curr_stat).checked) {
                resultMG(1, curr_stat);
            } else {
                resultMG(0, curr_stat);
            }
            stopit = 1;
        } else {
            simb = para_input[pos];
            for (var k = 0; k < alfabeto.length && alfabeto[k] != simb; k++);
            if (k == alfabeto.length) {
                resultMG(2, curr_stat);
                stopit = 1;
            } else {
                var val = document.getElementById("X_" + curr_stat + "," + k);
                var strUser = val.options[val.selectedIndex].value;
                if (strUser == "NaN") {
                    maquina_encravada2(simb, curr_stat);
                    stopit = 1;
                } else {
                    prev_stat = curr_stat;
                    curr_stat = parseInt(strUser.substring(1));
                }
            }
        }
    }
    var resultado = [result, mostra_input_consumido("charge", pos, para_input)];
    return resultado;
}
```

Figura 32 - Função simular_1()

O resultado é apresentado sobre a forma de tabela, em que cada palavra é uma linha dividida em colunas, em que a primeira coluna contém a palavra testada, a segunda coluna o resultado e a terceira o input consumido, neste caso em que posição terminou a leitura para o resultado obtido.

Os resultados possíveis são sucesso com identificação de qual o estado de aceitação onde foi alcançado, falha com identificação do estado onde terminou a palavra, Encravamento da Máquina por não existir transição de determinado símbolo quando estávamos em determinado estado e Símbolo não reconhecido.

A tabela 5 apresenta o teste de algumas palavras para o autómato apresentado na *figura 4*.

input	resultado	input consumido
000	**Sucesso!** Estado final alcançado q0	000[]
001	**Sucesso!** Estado final alcançado q1	001[]
010	**Sucesso!** Estado final alcançado q0	010[]
011	**Falha.** Alcançou o estado não final q2	011[]
100	**Sucesso!** Estado final alcançado q0	100[]
101	**Sucesso!** Estado final alcançado q1	101[]
110	**Falha.** Alcançou o estado não final q2	110[]
111	**Falha.** Alcançou o estado não final q2	111[]
aa	Símbolo não reconhecido: **a** não está alfabeto...	[a]a
ab	Símbolo não reconhecido: **a** não está alfabeto...	[a]b
ba	Símbolo não reconhecido: **b** não está alfabeto...	[b]a
bb	Símbolo não reconhecido: **b** não está alfabeto...	[b]b
10b	Símbolo não reconhecido: **b** não está alfabeto...	10[b]

Tabela 5 - Tabela de Simulação

B1.5 – Interface Gráfica

Figura 33 - Logótipo UAbALL

Para a interface gráfica, uma das necessidades percecionadas foi para a existência de um logotipo que representasse este laboratório. Foi desta análise que surgiu a construção do logo UAbALL, construído de raiz a pensar na modelação de autómatos, e contendo de raiz o azul, que é uma das principais cores da Universidade Aberta. Apresentado sobre um conjunto de roldanas, que pretendem representar os estados do autómato, ligadas por uma fita que vai sendo lido pela máquina.

Como estamos perante uma aplicação Web o controlo gráfico da apresentação é realizada através do mecanismo CSS (Cascading Style Shets) que adiciona os diferentes estilos. De seguida são apresentadas as principais aplicações realizadas com a folha de estilo externa *style.css* (disponível no anexo 5).

Através do seletor de id #footer afixamos este no fundo, ocupando toda a largura disponível, com um fundo a negro e letras a branco, com exeção da versão que fica a vermelho, o tamanho da fonte é 0.75 do elemento corrente e alinhado ao centro, foi fixada a cor do link, quando visitado, para permanecer inalterado.

```
#footer {
    position: fixed;
    left: 0;
    bottom: 0;
    width: 100%;
    background-color: #333333;
    font-size: 0.75em;
    color: #ffffff;
    text-align: center;
}
#footer a:link,
a:visited {
    color: #ffffff;
}

#footer small {
    color: #ff0000;
}
```

Figura 34 - #Footer por style.css

O menu é fixado no topo através do seletor de id #menu, o qual é utilizado também para a definição das cores utilizadas, o tamanho da fonte, a colocação das bandeiras de idioma, a cor de fundo ao passar com o rato por cima das opções, ou mesmo a colocação de um visto ao passar pela seleção de idioma.

Com o seletor de id #corpo fixamos a largura da imagem do ecrã de boas vindas para 80%, assim como ajustamos os textos para um alinhamento justificado, com um tamanho de 0.90 do elemento corrente, bem como uma margem esquerda e direita.

Com o #mycanvas definimos a posição e dimensão do container que receberá e projetará a cena criada com o *Three.js*.

```
#menu {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333333;
    font-size: 0.95em;
    font-weight: bold;
}

#menu li {
    float: left;
}

#menu li a {
    display: block;
    color: #ffffff;
    text-align: center;
    padding: 17px 10px;
    text-decoration: none;
}

#menu li a:hover {
    background-color: #999999;
}
...
#menu li.prt {
    margin-right: 5px;
    margin-top: 7px;
    float: right;
    width: 50px;
    background: url("./imgs/ptflag.png");
    background-position: center center;
    background-repeat: no-repeat;
}
#menu li.esp a:hover,
#menu li.eng a:hover,
#menu li.prt a:hover {
    background: url("./imgs/check.png");
}
```

Figura 35 - #menu por style.css

Para o simulador DFA, foram ativas algumas propriedades para os diferentes seletores, mais concretamente aplicação de 0.7 de tamanho aos elementos com seletor **, definição dos botões de construção com fonte de 1.1em e fundo azul, definição dos botões limpar e simular da tabela de simulação e o tipo de letra e alinhamento da tabela de simulação. Todas as opções tomadas podem ser consultadas no respetivo anexo, nesta fase as opções não acrescentam alterações diferentes das técnicas já debatidas anteriormente, pelo que não existe necessidade de entrar em mais detalhe sobre estas.

Para o DFA foi definida uma nova função para o controlo visual dos componentes que são apresentados, estando ocultos, quando esta função é chamada (figura 36).

Esta função utiliza a propriedade *display*[16] de CSS que especifica o tipo de renderização

```
function inOutDiv() {
    // ativa div tabela de testes
    document.getElementById("charge").style.display = "block";
    // carrega base de testes das variaveis globais e introduzidas
    document.getElementById("results").innerHTML =
        '<textarea id="base_tests" rows=15 cols=50>' + base_tests + "</textarea>";
    // ativa div com os resultados da tabela de testes
    document.getElementById("results").style.display = "block";
    // esconde ficheiro carregar, deixa fazer sentido nesta fase
    document.getElementById("file").style.display = "none";
}
```

Figura 36 - Função inOutDiv()

utilizada para um elemento, onde quando utilizado o valor *none* desativamos a exibição, não só deste elemento, mas também de todos os descendentes e o documento é renderizado como se não existisse esse elemento. Para ativar utilizamos o valor b*lock*.

B1.5.1 – Criação de Gráfico e Simulação Passo-a-Passo

Para a criação do gráfico iniciamos o processo pela construção do exemplo discutindo na figura 4, no editor online do *Graphviz*[17]. A partir desta construção ficamos com a definição da informação que precisamos de passar para o construtor do gráfico no processo inicial, mas também do efeito que pretendemos a cada passo da simulação para conseguirmos um efeito como o apresentado na figura 10Figura 10 - Simulação DFA .

Esta análise é visível através das figuras 37 e 38, destas conseguimos observar os dados necessário para a construção do grafo do autómato, assim como das alterações necessárias para que cada passo da simulação seja visível e síncrona com o momento (passo).

[16] https://www.w3schools.com/jsref/prop_style_display.asp
[17] http://magjac.com/graphviz-visual-editor/

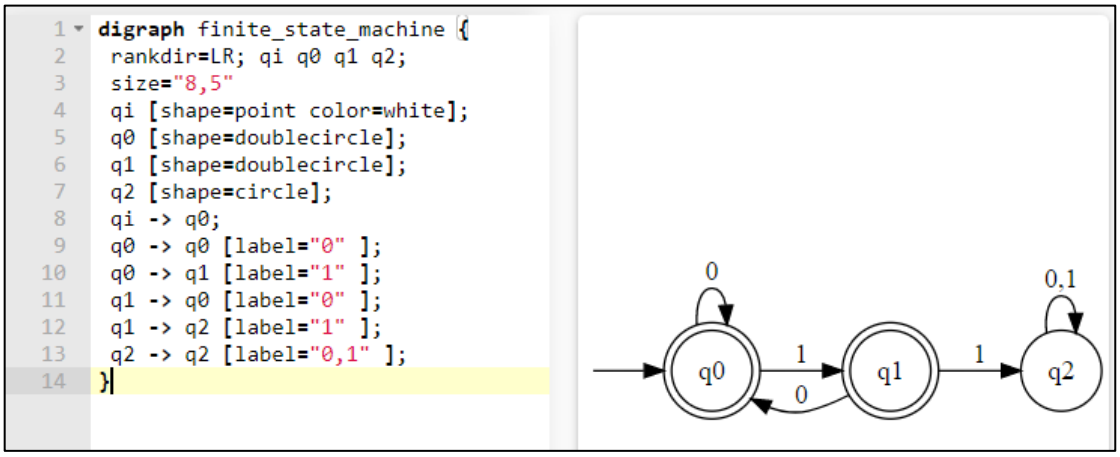


Figura 37 - Simulador Graphviz

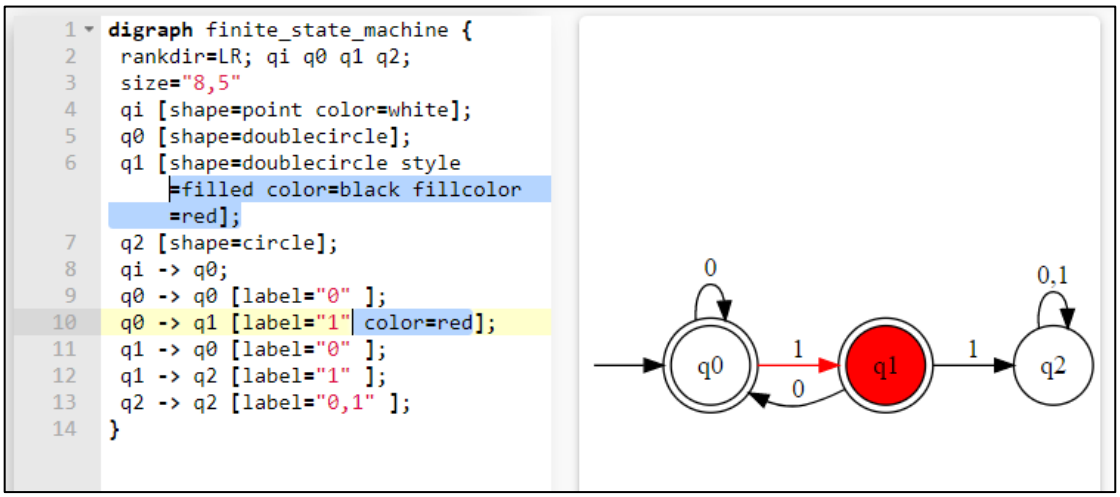


Figura 38 - Simulador Graphviz passo 1[1]1

Para a construção do gráfico temos a função apresentada na figura 39. Começamos pela recolha o alfabeto e o número de estados, verificamos se já estamos em simulação, se sim vamos marcar o passo no gráfico e mostrar qual o input consumido, sendo que este poderá ser um estado final de aceitação ou um contrário e se estivermos no último caracter da palavra a simular deveremos apresentar o respetivo resultado. Sendo esta parte da simulação passo a passo, deixamos o detalhe para mais tarde.

Iniciamos então a construção do grafo *viz*, para tal vamos guardar numa *string*, neste caso *cod_viz,* aos diferentes componentes começando pelo *rankdir* que com atribuição "LR" identifica o tipo de output de construção do grafo, ou seja da esquerda para a direita. Por cada estado criamos um nó e iniciamos um ponto "qi" branco, para não ficar visível, mas permitindo criar o apontador para o nó inicial. Na fase seguinte para cada estado colocamos os atributos de forma *(shape=circle*), sendo que caso seja um estado

```
function create_graph_desc() {
    alfabeto = document.getElementById("symbols").value;
    nEstados = parseInt(document.getElementById("nEstados").value);
    stopit = 0;
    if (simulate == 1) {
        primeiro = 0;
        if (prev_stat == -1 && curr_stat == -1) {
            curr_stat = estadoInicial;
            primeiro = 1;
        }
        pos_input++;
        mostra_input_consumido("passo", pos_input, input);
        if (pos_input == input.length) {
            messagem_final();
            stopit = 1;
        } else if (primeiro != 1) {
            simb = input[pos_input];
            for (k = 0; k < alfabeto.length && alfabeto[k] != simb; k++);
            if (k == alfabeto.length) {
                mensagem_erro_sim(simb);
                stopit = 1;
            } else {
                var val = document.getElementById("X_" + curr_stat + "," + k);
                var strUser = val.options[val.selectedIndex].value;
                if (strUser == "NaN") {
                    maquina_encravada();
                    stopit = 1;
                } else {
                    prev_stat = curr_stat;
                    curr_stat = parseInt(strUser.substring(1));
                }
            }
        }
    }
    cod_viz = "digraph finite_state_machine {\n rankdir=LR; qi";
    for (i = 0; i < nEstados; i++) cod_viz = cod_viz + " q" + i;
    cod_viz = cod_viz + ';\n size="8,5"\n';
    cod_viz = cod_viz + " qi [shape=point color=white];\n";
    for (i = 0; i < nEstados; i++) {
        if (document.getElementById("Final" + i).checked) {
            if (curr_stat == i && simulate == 1)
                cod_viz =
                    cod_viz +
                    " q" +
                    i +
                    " [shape=doublecircle style=filled color=black fillcolor=red];\n";
            else cod_viz = cod_viz + " q" + i + " [shape=doublecircle];\n";
        } else {
            ...
```

```
            if (curr_stat == i && simulate == 1)
                cod_viz = cod_viz + " q" +  i +
                    " [shape=circle style=filled color=black fillcolor=red];\n";
            else cod_viz = cod_viz + " q" + i + " [shape=circle];\n";
        }
    }
    var lig = [];
    for (i = 0; i < nEstados; i++) {
        lig[i] = [];
        for (j = 0; j < nEstados; j++) {
            lig[i][j] = "";
        }
        for (j = 0; j < alfabeto.length; j++) {
            var val = document.getElementById("X_" + i + "," + j);
            var strUser = val.options[val.selectedIndex].value;
            to_state = parseInt(strUser.substring(1));
            lig[i][to_state] = lig[i][to_state] + alfabeto[j];
        }
    }
    cod_viz = cod_viz + " qi -> q" + estadoInicial + ";\n";
    for (i = 0; i < nEstados; i++) {
        for (j = 0; j < nEstados; j++) {
            if (lig[i][j] != "") {
                if (simulate == 1 && prev_stat == i && curr_stat == j)
                    cod_viz = cod_viz +  " q" + i + " -> q" + j + ' [label="' +
                        lig[i][j].split("").join() + '" color=red];\n';
                else
                    cod_viz = cod_viz +  " q" + i + " -> q" + j + ' [label="' +
                        lig[i][j].split("").join() + '" ];\n';
            }
        }
    }
    cod_viz = cod_viz + "}\n";
    document.getElementById("graphviz_svg_div").innerHTML = "";
    var svgText = Viz(cod_viz, "svg");
    document.getElementById("graphviz_svg_div").innerHTML = "<hr>" + svgText;
    document.getElementById("simula_viz").style.display = "block";
    if (stopit == 1) {
        simulate = 0;
        pos_input = -2;
        prev_stat = -1;
        curr_stat = -1;
    }
}
```

Figura 39 - Função create_graph_desc()

de aceitação utilizamos a forma "*shape=doublecircle*". Por fim para cada estado temos que criar os nós de ligação na forma ***q0 -> q1 [label="0,1"]*** , assim que corridos todos os estados podemos fechar a nossa string de construção do gráfico e enviar este para o construtor do grafo.

Passo-a-passo

Na simulação passo-a-passo, a cada um dos passos chamamos novamente todo o processo anterior, com a identificação do passo [indexante] que estamos a testar, e esse será preenchido a vermelho, como o detalhe assinalado na figura 38Figura 38 - Simulador Graphviz passo 1[1]1. Este processo é iniciado com a chamada da função *percorre()* cada vez que o botão "Passo" é pressionado no laboratório, esta função ativa o controlador de simulação iniciada, caso seja a primeira chamada, imprime o resultado do passo dado, caso tenhamos chegado ao fim da palavra reinicia os contadores das variáveis e volta a chamar a função de criação do gráfico.

Para o leitor poderá ter ficado a ideia de que operação é muito simples do ponto de vista computacional, contudo é importante compreender que o processamento não é assim tão imediato, por exemplo quando aparece "0,1" no mesmo label tem de haver um pré-processamento anterior. Percebemos que não é uma coisa linear passar da tabela para o autómato, bem como fazer a simulação.

B2. TESTES

Para a gestão da qualidade do desempenho serão produzidos testes recorrendo à plataforma GTmetrix, onde foram comparadas versões para parâmetros não relacionados com o alojamento da aplicação.

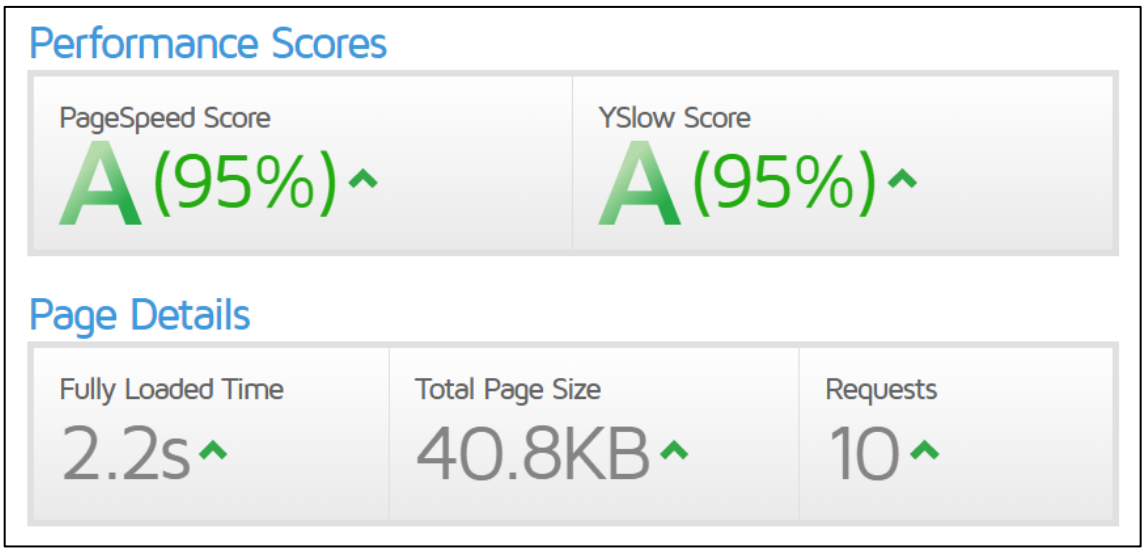


Figura 40 - GTmetrix 1ª Versão Online 04/05/2020

A figura *40* é a avaliação da *webpages* na sua primeira versão, enquanto que a figura 41 é a mesma avaliação para uma segunda versão, onde se obteve um ganho de carregamento

de quase um segundo, com uma redução de 0,6 KB no tamanho da página. Esta otimização foi conseguida com a adaptação dos blocos carregados por JavaScript para blocos de texto não indentados. Este modelo de parametrização será aplicado a todos os blocos com estas características, voltando a realizar novos testes e respetivas implementações de melhoria de desempenho, aquando da fase de Manutenção, no Capítulo 4.

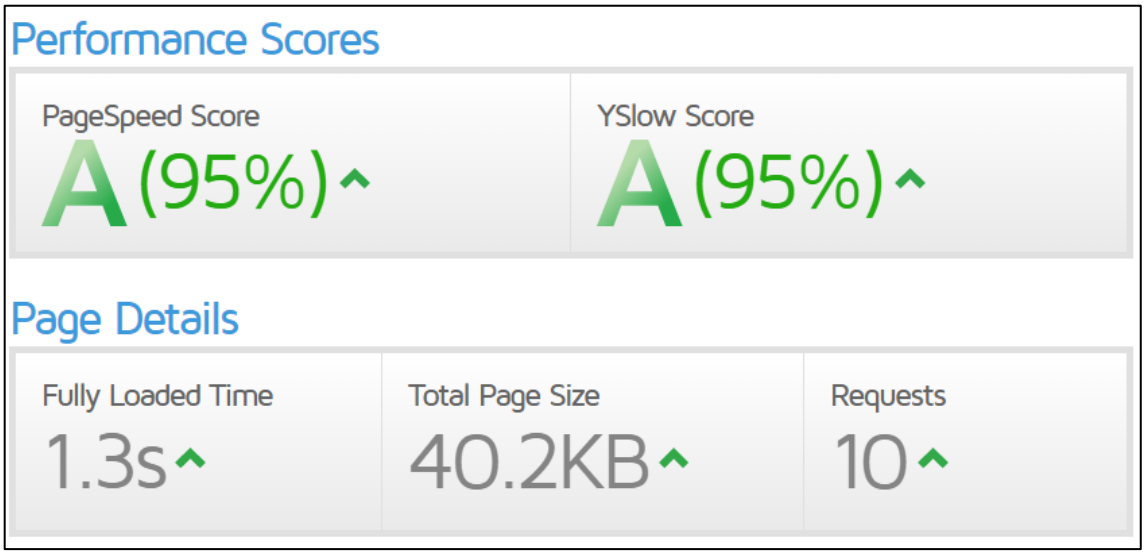


Figura 41 - GTmetrix 2ª Versão Online 05/05/2020

Para o utilizador a versão disponibilizada encontra-se identificada no Footer da aplicação, enquanto para o programador está presente no cabeçalho dos ficheiros html e JavaScript, a literatura deverá manter o registo das alterações a cada versão, utilizando para tal o próximo capítulo, no momento do encerramento deste capítulo a versão para o utilizador é a 0.9 e a técnica a 0.92.

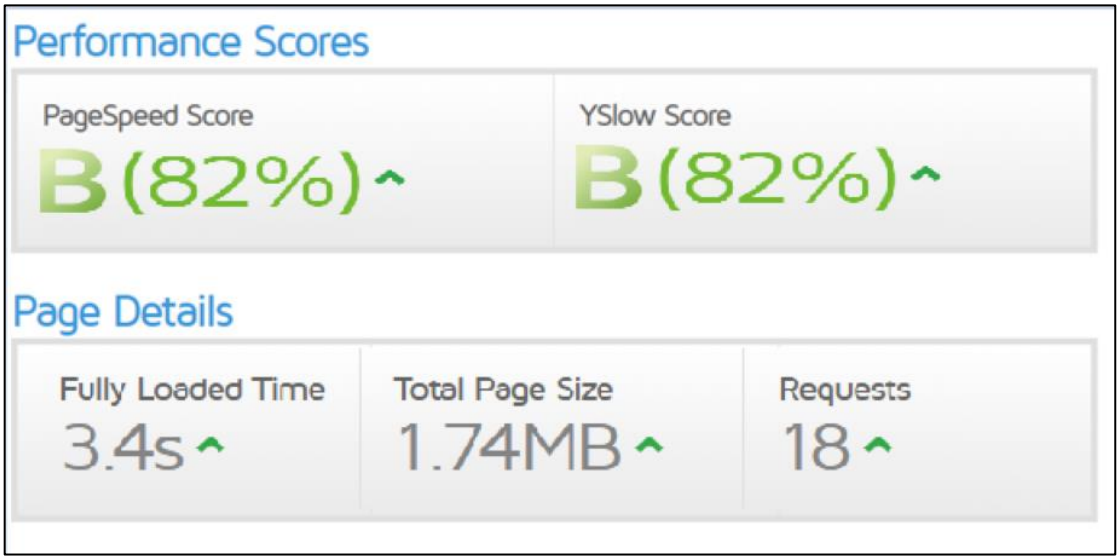


Figura 42 - GTmetrix Versão Online 28/08/2020

Neste teste já estava presente todo o conteúdo e temos um decréscimo de desempenho, sendo que entre as medidas necessárias para melhor este desempenho, estão a compressão dos ficheiros JavaScript e html, mas também a utilização de CDN (Content Delivery Network) para os componentes estáticos, neste caso os ficheiros JavaScript e o de CSS.

Esta questão deverá ser implementada aquando da conclusão do restante laboratório, dado que os ficheiros em causa serão alvo de correções e de transformações.

BUG leitura de ficheiro, no Chrome e Edge

Durante os primeiros testes, antes da disponibilização de uma versão 0.99 para um teste mais alargado com vários utilizadores, foi detetada uma falha no processamento de leitura de ficheiros nos browsers *Chrome* e *Edge*, a partir da versão 85 destes.

A leitura de ficheiros havia sido produzida, logo na primeira fase, dado ser muito útil para testar várias variáveis sem ter necessidade de estar sempre a colocar opções distintas no simulador, pelo que como esteve sempre funcional, foi facilmente apontada como sendo uma questão relacionada com os browsers. Partindo para a verificação do que poderia existir em comum entre ambos, denota-se a utilização por ambos do *Chromium*, mas como o *Edge* já utiliza desde janeiro 2020, ainda não era suficiente para a despistagem do que poderia estar a acontecer para que não fossem carregados os ficheiros guardados previamente.

Pensado que o processamento era sempre realizado pela ordem de chamada, foi descurada a possibilidade de leitura em paralelo, que poderá mudar em conformidade com o navegador utilizado. Existia a necessidade de garantir que primeiro ocorria a leitura e só depois o restante, para tal a função *ler_UAbALL* passou apenas a chamar a leitura do ficheiro, passando uma função que será chamada quando a leitura for feita, o buffer deixou de ser necessário, visto que depois de ler o ficheiro, passa por parâmetro o conteúdo lido. Estas alterações são destacadas nas figuras *43 e 44*.

```
//-- Carregamento de ficheiro --
function ler_UAbALL() {
    loadFileAsText(ler_UAbAllCont);
}
function ler_UAbAllCont(fileContent)
{
    // loadFileAsText();
        ...
    inOutDiv();
}
```

Figura 43 - Versão 1.0 da Função ler_UAbALL()

```
function loadFileAsText(callBack) {
    var fileToLoad = document.getElementById("fileToLoad").files[0];
    var fileReader = new FileReader();
    fileReader.onload = function (fileLoadedEvent) {
        callBack(fileLoadedEvent.target.result);
    };
    fileReader.readAsText(fileToLoad, "UTF-8");
}
```

Figura 44 - Versão 1.0 da Função loadFileAsText()

Capítulo 4

C- Manutenção

Neste capítulo abordaremos a primeira implementação em ambiente real para a qual foram convidados vários estudantes da Universidade Aberta a testarem e avaliarem o laboratório disponibilizado no Moodle, no Clube de Informática.

A primeira secção disponibiliza um manual de utilização para o laboratório, sendo as secções seguintes para o resultado dos inquéritos ao grupo de testes e por fim algumas correções resultantes das questões deixadas por este grupo, e discussão das que não foram implementadas.

C1. DISPONIBILIZAÇÃO VERSÃO 1.0

Para que o utilizador final possa ter a melhor experiência com o software é apresentado de seguida um manual de utilização, com uma descrição do funcionamento, para obter um melhor rendimento da utilização do laboratório e evitar possíveis problemas. Este manual permite uma serie de possibilidades que o programa pode oferecer, servindo como um processo de compreensão para uma boa utilização do sistema.

Ecrã Principal

No ecrã apresentado na entrada do laboratório tem os disponíveis informações e operações para o utilizador, a figura 45 assinala as áreas que passamos a descrever com a respetiva referência.

1. Esta é área da aplicação que apresenta a funcionalidade para o multilingue, onde ao passar com o rato por cima das bandeiras é apresentado um visto que confirma a possibilidade de seleção, sendo que nesta versão estão disponíveis versões em Português, Inglês e Espanhol.

2. Aqui temos o menu de opções de autómatos e o regresso ao ecrã inicial com o universal botão “home”. Nesta versão está construído o simulador DFA, os restantes contêm o resumo científico do respetivo autómato e uma animação em Three.js com informação de que está em construção e que será disponibilizado o simulador em futuras versões.

3. Apresentamos aqui os recursos utilizados para a construção e operacionalização do programa. HTML5 - HyperText Markup Language – linguagem de marcação para construção de páginas web, CSS - Cascading Style Sheets – linguagem para adicionar estilos ao modo como são apresentados os documentos HTML, JavaScript – linguagem de programação interpretada de alto nível, que

Figura 45 - Ecrã Principal UAbALL

conjuntamente com HTML e CSS perfaz as três principais tecnologias da World Wide Web, Moodle Versão 3.3.9+ da Universidade Aberta, no espaço do Clube de Informática, jQuery - uma biblioteca JavaScript desenvolvida para simplificar os scripts interpretados no navegador do utilizador, Viz.js - um cliente de Graphviz em JavaScript.

4. Apresentação do autor com hiperlink para a página web e versão atualizada do UAbALL.

DFA

Com a seleção, no menu, da opção DFA é nos apresentado o ecrã da figura 46. Tendo neste momento duas opções, o carregamento de um ficheiro previamente guardado, ou a construção do primeiro autómato.

Para a construção do primeiro autómato, tomamos como referência o descritivo científico da função e selecionamos o número de estados pretendidos e qual o alfabeto, neste caso os símbolos seguidos e sem qualquer separação. Após estes primeiros passos,

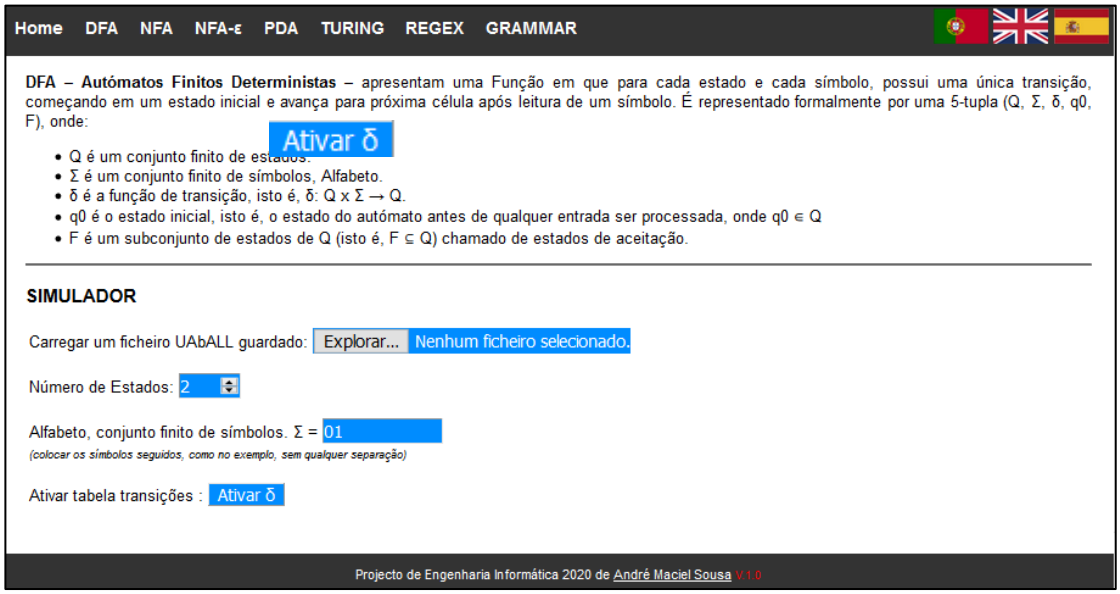


Figura 46 - DFA - Ecrã inicial

utilizamos o botão para ficarmos com a base de construção da tabela de transições. Temos agora de selecionar qual o estado inicial (qo), quais os estados de aceitação (Final) e para cada símbolo qual a transição que será realizada. **Importante** referir que caso seja deixada uma transição vazia ("*NaN*") para um dos símbolos, caso uma das palavras a testar utilize essa transição, iremos provocar um encravamento da máquina.

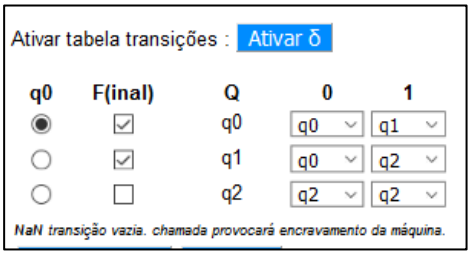


Figura 47 - DFA - Tabela Transições

Aquando da ativação da tabela de transições fica disponível a tabela de simulações, que permite testar um conjunto de palavras, sendo que por defeito é já apresentado um primeiro conjunto, para acrescentar bastará adicionar no primeiro espaço vazio, separando sempre com um *enter* cada palavra, mas é possível apagar todas as apresentadas e colocar novas.

Para simular na tabela, necessita de premir o botão Simular, e será apresentado o resultado de cada palavra com três colunas, que representam, por esta ordem, qual a palavra testada (*input*), qual o resultado do teste e para este resultado qual foi o último input consumido. Para nova simulação é necessário premir opção Limpar.

Para guardar o autómato e a tabela de simulação num ficheiro, deverá ser utilizado o botão Guardar, que abrirá uma janela onde deverá colocar o nome que pretende para o ficheiro a guardar e escolher a localização no seu computador, o comportamento desta opção difere de sistema operativo e browser, uma vez que por exemplo o *Edge* guarda em automático na pasta de transferência, não questionando para a localização pretendida.

Para o carregamento de um ficheiro previamente guardado deverá ser premido o botão de Explorar... escolher o local onde está guardado o ficheiro, selecionar o mesmo e abrir, o sistema apresenta de seguida uma mensagem de sucesso.

O laboratório tem uma componente visual, que permite uma interação com o autómato, com visualização do seu funcionamento. A construção é realizada quando pressionado o botão Construir DFA é disponibilizado o gráfico e as opções passo-a-passo.

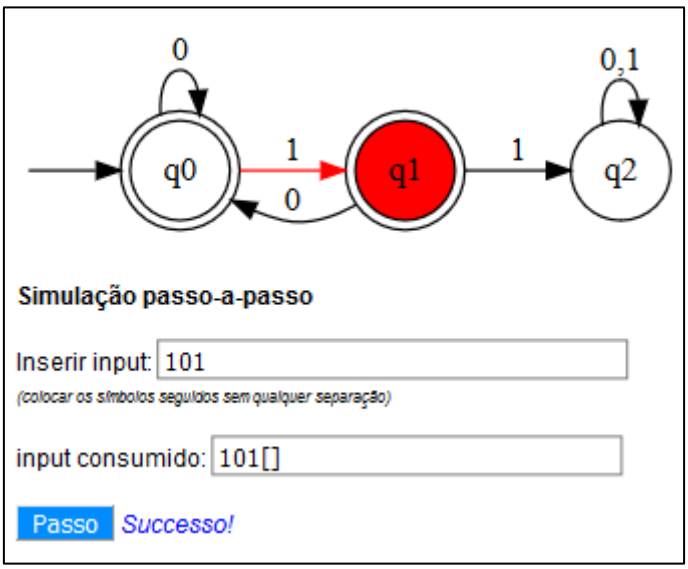


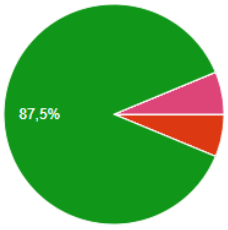
Figura 48 - Exemplo DFA Simulação Passo-a-Passo

Na utilização deste componente deverá ser carregada a palavra a testar, sem qualquer separação entres os símbolos, e de seguida pressionar o botão Passo que a cada vez que é pressionado mostra no gráfico o passo e marca o input consumido na respetiva caixa, quando termina a verificação é apresentada a mensagem do resultado.

C2. RESULTADOS INQUÉRITOS

Com a disponibilização da versão 1.0 na plataforma Moodle, foram convidados os estudantes da UAb, presentes no Clube de Informática, a testarem o UAbALL e responderem a um inquérito de avaliação e comunicação de erros e sugestões de melhoria. Ao momento de encerramento deste documento haviam respondido ao inquérito 36 estudantes, cujos resultados são apresentados de seguida.

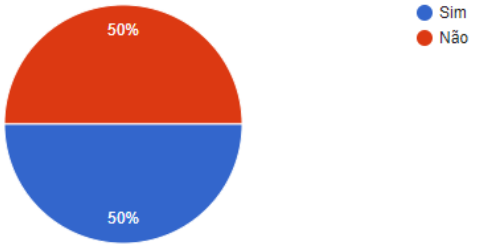
1. Qual o Curso que frequenta?



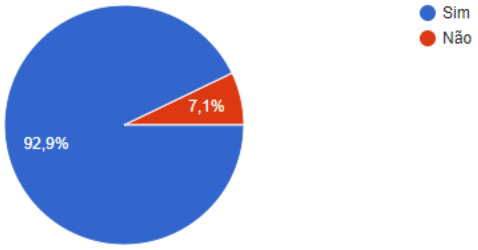
Licenciatura Engenharia informática 87.5%
Licenciatura em Gestão 6.25%
Licenciatura em Ciências Sociais 6.25%

2. Para os estudantes da Licenciatura em Engenharia Informática:

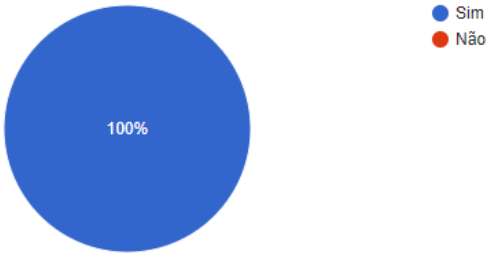
    a. Já frequentou a Unidade Curricular 21078- Linguagens e Computação?



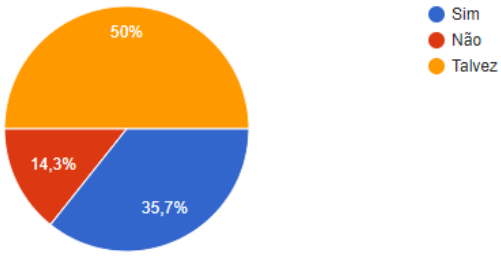
    b. Considere importante a existência de laboratórios práticos integrados na plataforma?



    c. Considera que este projeto poderá ser uma mais valia para a UC 21078- linguagens e computação?

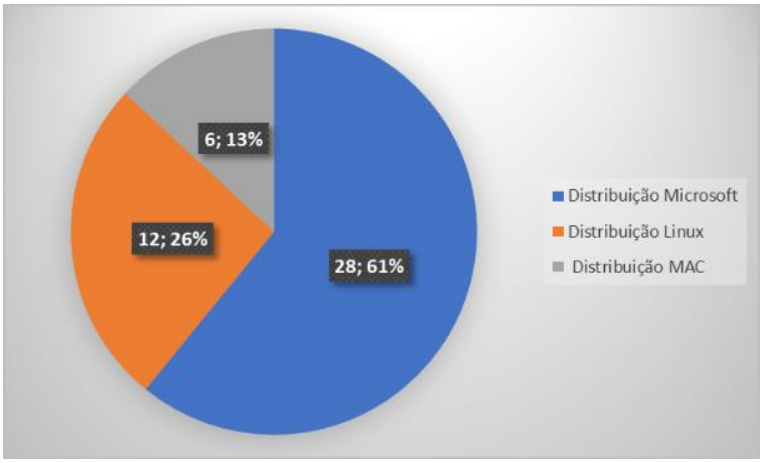


    d. Consideraria participar na produção de um dos módulos restantes?

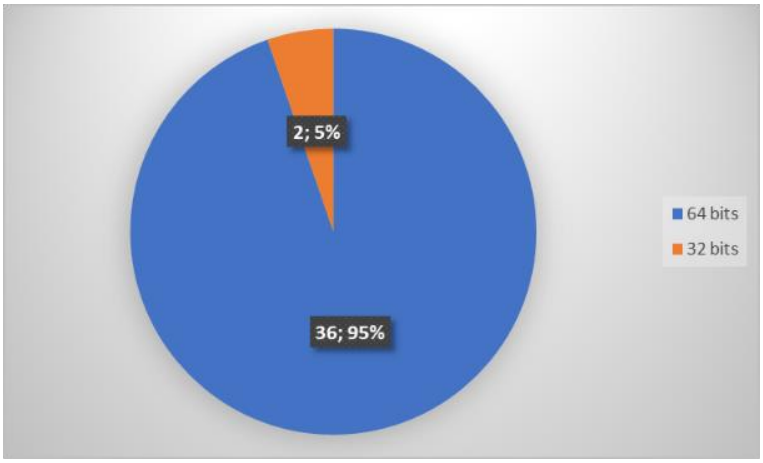


3. Ambiente Utilizado nos Testes:

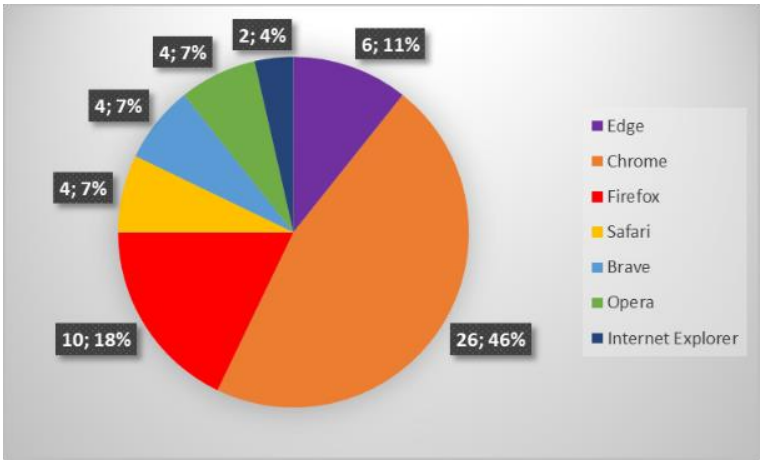
   a. Sistema Operativo?



   b. Tipo de Sistema?

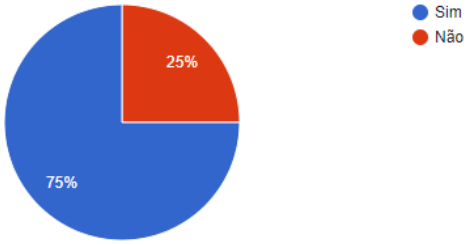


   c. Navegadores Utilizados?

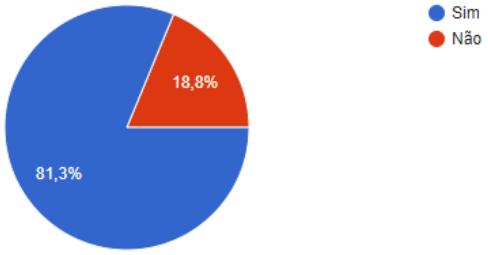


4. Opções utilizadas no UAbALL:

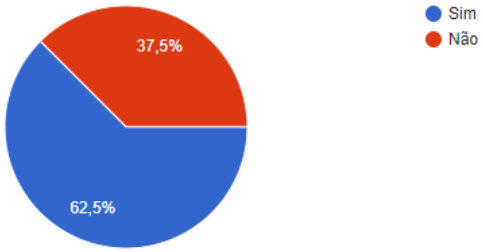
   a. Utilizou opção Multilingue?



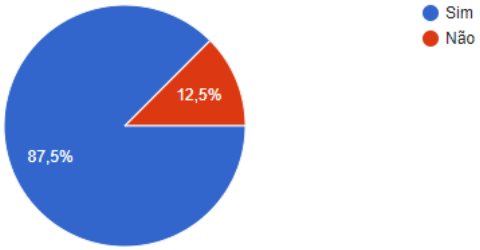
   b. Utilizou a construção manual do DFA?



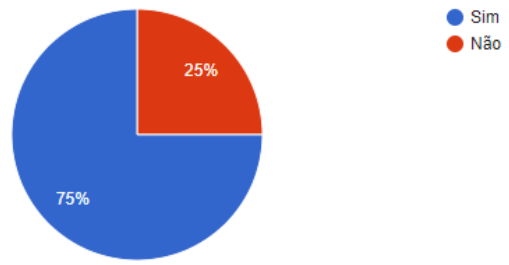
   c. Utilizou Guardar Autómato?



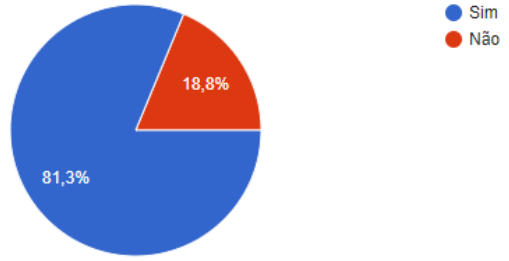
   d. Utilizou Construir DFA para o grafo?



e. Simulou passo-a-passo?

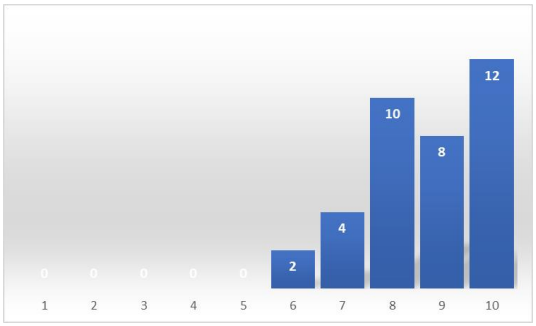


f. utilizou a tabela de Simulação?

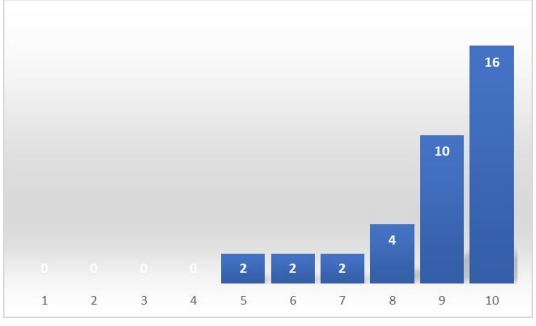


5. Avaliação do Laboratório numa escala de 1 a 10, onde 1 é fraco e 10 é excelente:

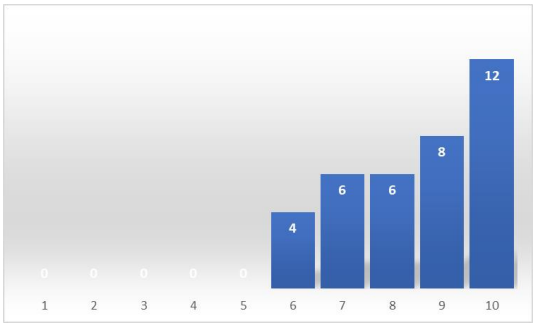
a. Avaliação Geral?



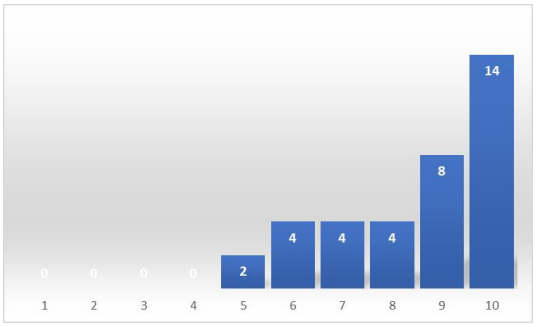
b. Opção Multilingue?



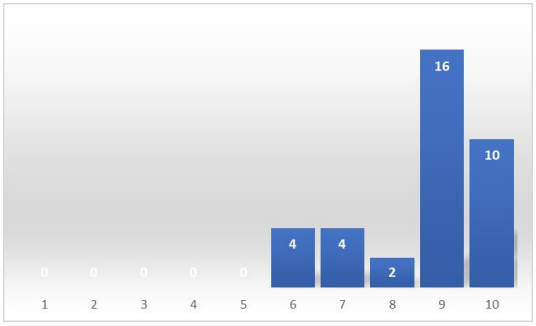
c. Construir DFA?



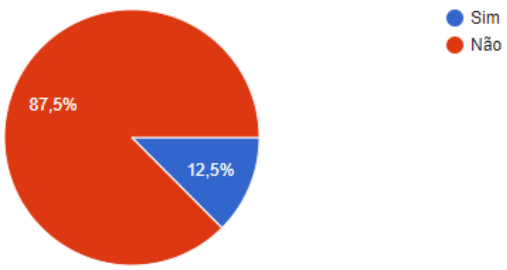
d. Simulação passo-a-passo?



e. Tabela Simulação?



6. Encontrou falhas ou tem propostas melhoria?



Destes resultados além da experiência de ter tido colegas de outras licenciaturas a experimentar este laboratório, o que não é muito habitual, logo muito positivo, acresce

a capacidade de ter sido possível que este projeto tivesse sido testado em diferentes ambientes.

De destacar que os estudantes da Licenciatura em Engenharia Informática, consideram importante para a Unidade Curricular de Linguagens e Computação a existência deste laboratório, assim como no modo geral a existência de projetos como este para outras unidades curriculares.

Todos os componentes foram testados e avaliados pelos estudantes, sendo que metade destes já frequentaram a Unidade Curricular, dando uma avaliação muito positiva ao projeto. As falhas e propostas de melhoria apresentadas serão discutidas na secção seguinte.

C3. RELEASE FIXES

No seguimento dos inquéritos de avaliação foram identificados dois temas, uma proposta de melhoria e uma falha no funcionamento.

A proposta de melhoria foi para que a tabela de simulação tivesse um **tamanho de letra maior**, uma vez que num ecrã de 11 polegada era pouco legível. Esta proposta foi colocada no código final, na folha de estilos para o seletor do id *simTable* passando de *0.8em* para *1em*.

A falha indica foi para "**Permite construir e guardar DFA sem estado final ou conjunto de estados finais**". De fato um autómato finito determinista apresenta para cada estado e símbolo uma única transição, começando num estado inicial e avançando por um conjunto finito de estados, sendo que deverão existir um conjunto finito de estados de aceitação que validam as palavras testadas. Contudo, este laboratório pretende acrescentar uma componente pedagógica, pelo que é permitida a simulação na condição indicada, para que o utilizador possa percecionar os diferentes comportamentos das opções tomadas. Além disso, o conjunto de estados finais de um autómato pode ser vazio.

Na Unidade Curricular de Linguagens e Computação é utilizado um simulador escrito em Java para experimentar tópicos da ciência da computação na área de linguagens formais e teoria de autómatos, o JFLAP[18]. Este simulador também permite contruir e guardar DFA sem estado final ou conjunto de estados finais, pelo mesmo princípio educacional. Com o UAbALL passamos a ganhar, comparativamente ao JFLAP, a integração em ambiente moodle e sem exigências de software do lado do utilizador, que não seja o que já utiliza para aceder ao Moodle. A construção dos autómatos é mais intuitiva, logo inclui menor período de adaptação.

[18] http://www.jflap.org/

Conclusão

Após a conclusão da fase de conceção o projeto esteve, praticamente, parado até ao final do mês de julho, motivado por questões de saúde, tendo sido retomado no princípio do mês de Agosto, tendo sido produzida uma correção do cronograma em conformidade com a tabela *6*.

B - Implementação	B1. Desenvolvimento B1.1 – Idiomas B1.2 – Menus B1.3 – Funcionalidades Globais B1.4 – Funcionalidades DFA B1.5 – Interface Gráfica	18-agosto-2020
	B2. Testes	28-agosto-2020
	R2. Relatório Final	31- agosto -2020
C – Manutenção	C1. Disponibilização Versão 1.0	01-setembro-2020
	C2. Resultados Inqueiro	08-setembro-2020
	C3. Release Fixes	08-setembro-2020
	P0. Preparação da Defesa do Projeto	23-setembro-2020

Tabela 6 - Cronograma Atualizado

O contínuo processo de registo de tomadas de decisão e descrição das funcionalidades ao longo do projeto foram fundamentais para que ao final de várias semanas sem trabalhar neste, tivesse sido possível voltar para a sua produção sem dificuldades acrescidas. Sendo, portanto, uma demonstração do cumprimento do objetivo: **Relatório técnico para trabalho futuro,** *produzir informação técnica suficiente para que o projeto possa ser adaptado a novas realidades tecnológicas e plataformas de distribuição*.

A plataforma apresentada reúne as condições necessárias para que venha a receber todas as funcionalidades apresentadas na secção *A2.2.1 – Definições básicas sobre a Teoria de Autómatos* com a futura inclusão dos autómatos NFA, NFA-ε, PDA, a maquina TURING e as representações estruturais REGEX e GRAMMAR, bem como a alteração do idioma de apresentação com base no idioma do utilizador, ou seleção realizada. Ficando, deste modo cumprido o objetivo de **Extensibilidade e adaptabilidade.**

Foi possível operacionalizar a base do Laboratório Virtual UAbALL, como também o seu primeiro módulo: SIMULAÇÃO DE AUTÓMATOS FINITOS DETERMINISTAS (DFA). Cumprindo deste modo o objetivo principal deste projeto, sendo este manuscrito uma ferramenta de apoio à produção dos restantes módulos, para que o UAbALL venha a ser um laboratório virtual de apoio à aprendizagem dos conceitos apresentados.

Todo o processo de testes e avaliação junto dos estudantes foi um desafio e uma mais valida para o futuro deste laboratório. O desafio de conseguir mobilizar estudantes em período de férias escolares foi uma ligeira dificuldade para a amostra que era pretendida. A mais valia para o futuro está no facto de termos hoje uma avaliação muito positiva deste projeto, além de uma aproximação de um conjunto de estudantes disponíveis para fazerem crescer este projeto.

Este projeto teria sido mais produtivo para a Licenciatura Engenharia Informática caso tivesse existido uma lógica de grupo de desenvolvimento para este laboratório. Não obstante, percebe-se que o facto de o projeto incluir a especificação da interface gráfica poderia causar dificuldades no arranque. Mas o facto de o projeto final ser de índole individual, o culminar de toda uma aprendizagem ao longo da licenciatura, não deve ser impeditivo de haver projetos que possam ser desenvolvidos em grupo, mesmo que a avaliação final seja individual. Fica assim, para memória futura, em consideração para as coordenações deste tipo de licenciaturas, que sejam pensados grupos de desenvolvimento para projetos desta envergadura. É fundamental que as Unidades Curriculares criem valor para a comunidade académica enquanto transmitem conhecimento aos discentes e colocam desafios para a evolução técnica e científica, não descurando a capacidade de organização e formação para o trabalho em grupo.

Bibliografia

Andersonmo. (18 de 03 de 2019). *FileReader*. Obtido em 15 de 08 de 2020, de MDN Web Docs: https://developer.mozilla.org/pt-BR/docs/Web/API/FileReader

Basic Overview of how LTI ® Works. (s.d.). Obtido em 20 de 04 de 2020, de IMS Global Learning Consortium: https://www.imsglobal.org/basic-overview-how-lti-works

CahMoraes. (06 de 02 de 2020). *Blob*. Obtido em 16 de 08 de 2020, de MDN Web Docs: https://developer.mozilla.org/pt-BR/docs/Web/API/Blob

Courouris, G., Dolimore, J., Kindberg, T., & Blair, G. (2012). *Distributed Systems - Concept and Design.* Addison Wesley LOngman.

Desconhecido. (s.d.). *Style display Property*. Obtido em 14 de 08 de 2020, de W3Schools.com: https://www.w3schools.com/jsref/prop_style_display.asp

Desconhecido. (s.d.). *Table insertRow() Method*. Obtido em 17 de 08 de 2020, de W3Schools.com: https://www.w3schools.com/jsref/met_table_insertrow.asp

Desconhecido. (s.d.). *TableRow insertCell() Method*. Obtido em 17 de 08 de 2020, de W3Schools.com: https://www.w3schools.com/jsref/met_tablerow_insertcell.asp

Desconhecido. (s.d.). *Window alert() Method*. Obtido em 14 de 08 de 2020, de W3Schools.com: https://www.w3schools.com/jsref/met_win_alert.asp

Dirksen, J. (2015). *Learning Three.js – the JavaScript 3D Library.* Packt Publishing Ltd.

Fscholz. (30 de 01 de 2019). *URL.createObjectURL()*. Obtido em 16 de 08 de 2020, de MDN Web Docs: https://developer.mozilla.org/pt-BR/docs/Web/API/URL/createObjectURl

Graphviz - Graph Visualization Software. (s.d.). Obtido em 19 de 04 de 2020, de Graphviz: https://www.graphviz.org/

Hopcroft, J., Motwani, R., & Ullman, J. (2007). *ntroduction to Automata Theory, Languages and Computation, 3rd.* Addison-Wesley.

HTML <iframe> Tag. (s.d.). Obtido em 21 de 04 de 2020, de w3schools: https://www.w3schools.com/tags/tag_iframe.asp

Iframe. (15 de 01 de 2018). Obtido em 20 de 04 de 2020, de Moodle: https://docs.moodle.org/38/en/Iframe

Menezes, P. (2000). *Linguagens Formais e Autômatos.* Sagra Luzzatto.

Miguel, A. (2015). *Gestão de Projetos de Software.* FCA.

Pereira, A., & Poupa, C. (2013). *Linguagens WEB* (5ª ed.). Lisboa: Edições Sílabo.

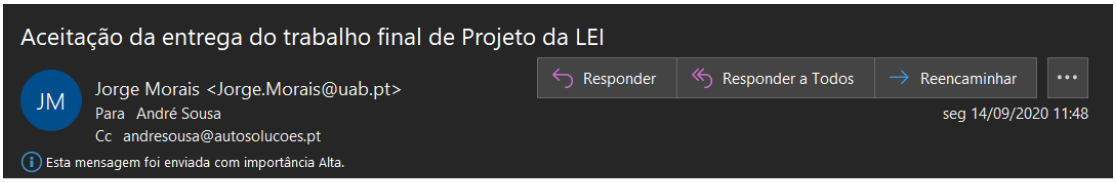
Pereira, J., Brisson, J., Coelho, A., Ferreira, A., & Gomes, M. (2018). *Introdução à Computação Gráfica.* FCA - Editora de Informática, Lda.

Sampaio, A. I. (20 de Dezembro de 2013). *Responsive Web Design*. Obtido de http://hdl.handle.net/1822/27902

Scalable Vector Graphics (SVG) 2. (2018). Obtido em 18 de 04 de 2020, de World Wide Web Consortium: https://www.w3.org/TR/SVG/

Wikipedia. (2020). *ISO 639-1*. Obtido em 09 de 04 de 2020, de Wikipedia: https://en.wikipedia.org/wiki/ISO_639-1

Aceitação Final Orientador



Aceitação da entrega do trabalho final de Projeto da LEI

JM

Jorge Morais <Jorge.Morais@uab.pt>

Para André Sousa

Cc andresousa@autosolucoes.pt

Esta mensagem foi enviada com importância Alta.

Responder | Responder a Todos | Reencaminhar | ...

seg 14/09/2020 11:48

Caro André,

tendo em conta a qualidade do trabalho, que considero excelente, e do respetivo relatório, dou o meu parecer positivo à entrega do mesmo, com vista à marcação da prova pública final.

Com os melhores cumprimentos,
Jorge Morais

Anexos

[1].INDEX.HTML

```
<!DOCTYPE html>
<html>
    <!--Versão 0.92 06/05/2020-->
    <head>
        <title>UAbALL</title>
        <meta charset="UTF-8" />
        <meta
            name="description"
            content="UAbALL - Automata Learning Lab | Um Simulador para o Ensino."/>
        <meta name="keywords" content="Automatos, gramaticas e Maquinas Touring" />
        <meta name="author" content="André Maciel Sousa" />
        <meta name="generator" content="Visual Studio Code" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <script type="text/javascript">
            function idioma() {
                switch (navigator.language.substring(0, 2)) {
                    case "en":
                        window.location.href = "./start.html?lang=EN";
                        break;
                    case "es":
                        window.location.href = "./start.html?lang=ES";
                        break;
                    default:
                        window.location.href = "./start.html?lang=PT";
                        break;
                }
            }
        </script>
    </head>
    <body onload="idioma();"></body>
</html>
```

[2]. LOADER.JS

```
// Versão 0.98 17/08/2020
// Variaveis Globais
var startedAnimation = false;
var lang; //idioma
var body; //corpo da pagina

// Definição de paginas e blocos
var menu = "./blocos/menu.txt";
var footer = "./blocos/footer.txt #";
var home = "./blocos/corpo.txt #";
var dfa = "./blocos/dfa.txt #";
var nfa = "./blocos/nfa.txt #";
var enfa = "./blocos/enfa.txt #";
var pda = "./blocos/pda.txt #";
var turing = "./blocos/turing.txt #";
var regex = "./blocos/regex.txt #";
var grammar = "./blocos/grammar.txt #";

// Funcao principal para carregamento conteudos
function main(body) {
    body = body || home; //se nulo atribui home

    $(document).ready(function () {
        if (!lang) {
            lang = $_GET("lang");
        }
        $("#nav").load(menu);
        $("#corpo").load(body + lang);
        // adicionar nova div para animacao "proximas versoes"
        $("#mycanvas").remove();
        if (body != home && body != dfa) {
            cria_newDiv(lang);
        }
```

```
        }
        $("#footer").load(footer + lang);
    });
}
// obter parametros da URL para ?
function $_GET(param) {
    var vars = {};
    window.location.href.replace(location.hash, "").replace(
        /[?]+([^=&]+)=?([^&]*)?/gi, // regexp
        function (m, key, value) {
            // callback
            vars[key] = value !== undefined ? value : "";
        }
    );
    if (param) {
        return vars[param] ? vars[param] : null;
    }
    return vars;
}
// adicionar nova div
function cria_newDiv(lang) {
    var novadiv = document.createElement("div");
    novadiv.setAttribute("id", "mycanvas");
    document.body.append(novadiv);
    underConstr(lang);
}

// Em construção animação Three.JS
function underConstr(lang) {
    init(lang);
    if (!startedAnimation) {
        startedAnimation = true;
        animate();
    }
}
```

[3]. START.HTML

```
<!DOCTYPE html>
<html>
    <!--Versão 0.98 17/08/2020-->
    <head>
        <title>UAbALL</title>
        <meta charset="UTF-8" />
        <meta
            name="description"
            content="UAbALL - Automata Learning Lab | Um Simulador para o Ensino."
        />
        <meta name="keywords" content="Automatos, gramaticas e Maquinas Touring" />
        <meta name="author" content="André Maciel Sousa" />
        <meta name="generator" content="Visual Studio Code" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <link href="./style.css" type="text/css" rel="stylesheet" />
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/109/three.js"></script>
        <script src="./js/loader.js"> </script>
        <script src="./js/underC.js"> </script>
        <script src="./js/viz.js"> </script>
        <script src="./js/dfa.js"> </script>
    </head>

    <body onload="main();">
        <!-- Menu -->
        <div id="nav"></div>
        <!--Corpo / Body-->
        <div id="corpo"></div>
        <!-- Footer -->
        <div id="footer"></div>

    </body>
</html>
```

[4]. MENU.TXT

```
<ul id="menu">
<li onclick="main(home);"><a href="#home">Home</a></li>
<li onclick="main(dfa);"><a href="#dfa">DFA</a></li>
<li onclick="main(nfa);"><a href="#nfa">NFA</a></li>
<li onclick="main(enfa);"><a href="#nfae">NFA-ε</a></li>
<li onclick="main(pda);"><a href="#pda">PDA</a></li>
<li onclick="main(turing);"><a href="#turing">TURING</a></li>
<li onclick="main(regex);"><a href="#regex">REGEX</a></li>
<li onclick="main(grammar);"><a href="#grammar">GRAMMAR</a></li>
<li class="esp"><a href="./start.html?lang=ES"></a></li>
<li class="eng"><a href="./start.html?lang=EN"></a></li>
<li class="prt"><a href="./start.html?lang=PT"></a></li>
</ul>
```

[5]. STYLE.CSS

```
body {
    margin: 0;
    font-family: sans-serif;
    font-size: 100%;
}

/************** Footer **************/
#footer {
    position: fixed;
    left: 0;
    bottom: 0;
    width: 100%;
    background-color: #333333;
    font-size: 0.75em;
    color: #ffffff;
    text-align: center;
}
#footer a:link,
a:visited {
    color: #ffffff;
}

#footer small {
    color: #ff0000;
}
/************** Menu Configurations **************/
#menu {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333333;
    font-size: 0.95em;
```

```
    font-weight: bold;
}
#menu li {
    float: left;
}
#menu li a {
    display: block;
    color: #ffffff;
    text-align: center;
    padding: 17px 10px;
    text-decoration: none;
}
#menu li a:hover {
    background-color: #999999;
}
#menu li.eng {
    margin-right: 5px;
    margin-top: 7px;
    float: right;
    width: 50px;
    background: url("./imgs/enflag.png");
    background-position: center center;
    background-repeat: no-repeat;
}
#menu li.esp {
    margin-right: 5px;
    margin-top: 7px;
    float: right;
    width: 50px;
    background: url("./imgs/esflag.png");
    background-position: center center;
    background-repeat: no-repeat;
}
```

```
#menu li.prt {
    margin-right: 5px;
    margin-top: 7px;
    float: right;
    width: 50px;
    background: url("./imgs/ptflag.png");
    background-position: center center;
    background-repeat: no-repeat;
}
#menu li.esp a:hover,
#menu li.eng a:hover,
#menu li.prt a:hover {
    background: url("./imgs/check.png");
}

/************** Start **************/
#corpo {
    text-align: justify;
    font-size: 0.9em;
    margin-left: 20px;
    margin-right: 20px;
}
#corpo img {
    width: 80%;
}

/************** Canvas THREE JS**************/
#mycanvas {
    text-align: center;
    position: absolute;
    width: 100%;
    height: 20%;
    margin-bottom: 15px;
    padding-bottom: 30px;
}
```

```
/************** Simulator DFA **************/

table,
th,
td {
    border: opx solid #oooooo;
    vertical-align: top;
}

em {
    font-size: o.7em;
}

/* botoes do DFA */
#fileToLoad,
#constroi_transitions,
#nEstados,
#symbols,
#simula_passo,
#create_graph_desc,
#save {
    font-size: 1.1em;
    color: white;
    background-color: #oo8cff;
}
/* tabela transições */
#transitions {
    text-align: center;
}
/* botao Simlar da tabela*/
#run_simul {
    font-size: 1em;
    color: white;
    background-color: #077004;
}
```

```
/* botao limpar da tabela*/
#limp_simul {
    font-size: 1em;
    color: white;
    background-color: #ff0000;
}
/* Tabela que recebe a div step-by-step */
#passos {
    margin-bottom: 15px;
    padding-bottom: 80px;
}

/* Div step-by-step */
#simula_viz {
    font-size: 0.9em;
}
/* Div step-by-step: mensagem resultado */
#resultado {
    font-size: 1em;
    color: blue;
}
/*Tabela de Simulação*/
#simTable {
    font-size: 1em;
    text-align: center;
    font-family: monospace;
}
```

[6]. UNDERC.JS

```
// Versão 0.91 19/08/2020
// ThreeJS
var scene, camera, renderer, container, atom;
function init(lang) {
    // referencia ao elemento que apresentara a scene
```

```
    container = document.querySelector("#mycanvas");
    // Criação do objeto cena que vai ser depois adicionado ao objeto render
    scene = new THREE.Scene();
    // Criação de camera virtual com projeção em perspectiva, que vai ser depois adicionada
ao objeto render
    camera = new THREE.PerspectiveCamera(
        45,
        container.clientWidth / container.clientHeight,
        0.1,
        1000
    );
    camera.position.set(0, 0, 60);
    camera.lookAt(0, 0, 0);
    // criacao dos objectos
    constroiObj();
    // coloca texto, em posição centrada e minima para versão telemovel
    // em conformidade com o idioma de navegação
    if (lang == "EN") {
        escreveTxt("Under construction...", "next versions.");
    } else if (lang == "ES") {
        escreveTxt("En construcción...", "próximas versiones.");
    } else {
        escreveTxt("Em construção...", "próximas versões.");
    }
    // Luz Ambiente
    luzAmbiente = new THREE.AmbientLight(0xffff00, 1);
    scene.add(luzAmbiente);
    // Renderizador
    renderer = new THREE.WebGLRenderer({ antialias: true, alpha: true });
    renderer.setSize(container.clientWidth, container.clientHeight);
    //definir proporção correta de pixels para o dispositivo em que está sendo executado
    renderer.setPixelRatio(window.devicePixelRatio);
    container.append(renderer.domElement);
}
```

```
// criacao dos objectos
function constroiObj() {
    atom = new THREE.Group();

    var geometry = new THREE.SphereBufferGeometry(10, 32, 32);
    var material = new THREE.MeshPhongMaterial({
        map: new THREE.TextureLoader().load("imgs/atom.jpg"),
        side: THREE.DoubleSide,
    });
    var aux = new THREE.Mesh(geometry, material);
    atom.add(aux);

    // Alguns clones
    for (i = 30; i < 90; i += 30) {
        var aux2 = aux.clone();
        aux2.position.set(-i, 0, -i / 5);
        atom.add(aux2);
        var aux3 = aux.clone();
        aux3.position.set(i, 0, -i / 5);
        atom.add(aux3);
    }

    aux = atom.clone();
    aux.rotation.z = Math.PI / 2;
    atom.add(aux);
    aux = atom.clone();
    aux.rotation.z = Math.PI / 4;
    atom.add(aux);

    scene.add(atom);
}
// Texto
function escreveTxt(txt1, txt2) {
    var loader = new THREE.FontLoader();
```

```
    loader.load("fonts/VW Head Office_Regular.json", function (font) {
        material = new THREE.MeshBasicMaterial({ color: 0x000000 });
        shapes = font.generateShapes(txt1, 2);
        geometry = new THREE.ShapeBufferGeometry(shapes);
        geometry.computeBoundingBox();
        texto1 = new THREE.Mesh(geometry, material);
        texto1.position.set(-38, 14, 20);
        scene.add(texto1);
        shapes = font.generateShapes(txt2, 2);
        geometry = new THREE.ShapeBufferGeometry(shapes);
        geometry.computeBoundingBox();
        texto2 = new THREE.Mesh(geometry, material);
        texto2.position.set(13, -15, 20);
        scene.add(texto2);
    });
}
// Resize
function onWindowResize() {
    // Se existir container, então estamos numa pagina com Three.Js
    if(container!=undefined){
    camera.aspect = container.clientWidth / container.clientHeight; // Para garantir que não
há distorção
    camera.updateProjectionMatrix(); // atualiza de seguida a matriz de projeção


    renderer.setSize(container.clientWidth, container.clientHeight);}
}
window.addEventListener("resize", onWindowResize, false);
// Adiciona efeito animado a cena
function animate() {
    atom.rotation.x += 0.002;
    atom.rotation.z += 0.002;
    atom.rotation.y += 0.002;
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
```

[7]. DFA.JS

```
// Versão 0.99 26/08/2020
// Variaveis globais
var nEstados; // numero de estados
var alfabeto; // alfabeto
var estadoInicial = 0; // Estado inicial
var simulate = 0; // simulação ainda não (re)iniciada
var prev_stat = -1; // indexante para estado anterior
var curr_stat = -1; // indexante para estado atual
var pos_input = -1; // indexante para posicao do input
var input = ""; // iniciação input
// base para testes para tabela simulacoes
var base_tests = "000\n001\n010\n011\n100\n101\n110\n111\naa\nab\nba\nbb\n10b";

// Construção tabela transições
function constroi_transitions() {
    // variaveis recolhidas do bloco dfa
    nEstados = parseInt(document.getElementById("nEstados").value);
    alfabeto = document.getElementById("symbols").value;

    //Contrução tabela trasições
    var tabela = document.getElementById("transitions");
    // (R)inicia tabela
    tabela.innerHTML = "";

    // Cria cabeçalho
    // primeira linha da tabela reservada para cabeçalho
    //https://www.w3schools.com/jsref/met_table_insertrow.asp [0=primeiro position || -1=last position]
    var row = tabela.insertRow(-1);
    //https://www.w3schools.com/jsref/met_tablerow_insertcell.asp [0=primeiro position || -1=last position]
    var cell = row.insertCell(-1);
    cell.innerHTML = "<b>q0</b>";
    var cell = row.insertCell(-1);
```

```
    cell.innerHTML = "<b>F(inal)</b>";
    cell = row.insertCell(-1);
    cell.innerHTML = "<b>Q</b>";
    for (var i = 0; i < alfabeto.length; i++) {
        cell = row.insertCell(-1);
        cell.innerHTML = "<b>" + alfabeto.charAt(i) + "</b>";
    }
    // para cada estado uma nova linha
    // o primeiro estado reservamos a possiblidade de ser o inicial
    for (var i = 0; i < nEstados; i++) {
        row = tabela.insertRow(-1);
        cell = row.insertCell(-1);
        if (i == 0)
            cell.innerHTML =
                '<input type="radio" id="initial" checked name="initial"
onchange="set_initial_state(' +
                i +
                ')" value="' +
                i +
                '">';
        else
            cell.innerHTML =
                '<input type="radio" id="initial" name="initial" onchange="set_initial_state(' +
                i +
                ')" value="' +
                i +
                '">';
        // o estado final poderá ser um ou mais a selecionar
        cell = row.insertCell(-1);
        cell.innerHTML = '<input type="checkbox"  id="Final' + i + '">';
        // identifica o estado de cada linha
        cell = row.insertCell(-1);
        cell.innerHTML = "q" + String(i);
        // para cada simbolo do alfabeto colocamos a possiblidade de transição
        for (var j = 0; j < alfabeto.length; j++) {
```

```
            str_select =
                '<select " id="X_' +
                i +
                "," +
                j +
                '">\n<option value="NaN">NaN</option>\n';
            // como estamnos com DFA todos os estados tem que ter uma transição,
            // para evitar erros, ate indicação contaria todos apontam para o ultimo estado
[v.098]
            /*  var mn = nEstados - 1;
                str_select =
                '<select " id="X_' +
                i +
                "," +
                j +
                '" >\n<option value="q' +
                mn +
                '">NaN</option>\n';  */
            for (var k = 0; k < nEstados; k++) {
                str_select =
                    str_select + '<option value="q' + k + '">q' + k + "</option>\n";
            }
            str_select = str_select + "</select>";
            cell = row.insertCell(-1);
            cell.innerHTML = str_select;
        }
    }
    // info para NaN
    row = tabela.insertRow(-1);
    cell = row.insertCell(-1);
    cell.colSpan = 5;
    if (lang == "EN") {
        cell.innerHTML =
            "<em><b>NaN</b>  empty transition. call will cause the machine to jam.</em>";
    } else if (lang == "ES") {
```

```
        cell.innerHTML =
            "<em><b>NaN</b> transición vacía. La llamada hará que la máquina se
atasque.</em>";
    } else {
        cell.innerHTML =
            "<em><b>NaN</b> transição vazia. chamada provocará encravamento da máquina.
</em>";
    }
    // Insere botoes de construção e guardar automato
    row = tabela.insertRow(-1);
    cell = row.insertCell(-1);
    cell.colSpan = 2;
    // butão para criar grafico
    if (lang == "EN") {
        cell.innerHTML =
            '<button id="create_graph_desc" onclick="create_graph_desc()">Build
DFA</button>';
    } else if (lang == "ES") {
        cell.innerHTML =
            '<button id="create_graph_desc" onclick="create_graph_desc()">Construir
DFA</button>';
    } else {
        cell.innerHTML =
            '<button id="create_graph_desc" onclick="create_graph_desc()">Construir
DFA</button>';
    }
    cell = row.insertCell(-1);
    // butão para salvar automato criado
    if (lang == "EN") {
        cell.innerHTML =
            '<button id="save" onclick="grava_UAbALL()">Save </button>';
    } else if (lang == "ES") {
        cell.innerHTML =
            '<button id="save" onclick="grava_UAbALL()">Salvar </button>';
    } else {
```

```
        cell.innerHTML =
            '<button id="save" onclick="grava_UAbALL()">Guardar </button>';
    }
    // ativação da div simulação passo a passo
    document.getElementById("simula_viz").style.display = "none";
    //-- Tratamento de Outputs Ativa/esconde Container--
    inOutDiv();
}

// Define atribuidor do identificador do estado inicial
function set_initial_state(ns) {
    estadoInicial = ns;
}
// criação do grafico com recurso ao viz.js
function create_graph_desc() {
    // variaveis recolhidas do bloco dfa
    alfabeto = document.getElementById("symbols").value;
    nEstados = parseInt(document.getElementById("nEstados").value);
    // variavel de controlo de simulação [ 1=Sim || 0=Não ]
    stopit = 0;
    // caso já tenha sido iniciada a simulação
    if (simulate == 1) {
        // primeiro no [ 1=Sim || 0=Não ]
        primeiro = 0;
        if (prev_stat == -1 && curr_stat == -1) {
            curr_stat = estadoInicial;
            primeiro = 1;
        }
        pos_input++;
        // Chama a funcao para identicação do input consumido
        mostra_input_consumido("passo", pos_input, input);
        // Se chegamos na ultima posição o imput verificamos In(Sucesso)
        if (pos_input == input.length) {
            messagem_final();
            stopit = 1;
```

```
            } else {
                // obtemos o estado seguinte da relação criada com o id
                // atualizamos as referencias para o anterior e seguinte
                /*      var val = document.getElementById("X_" + curr_stat + "," + k);
                var strUser = val.options[val.selectedIndex].value;
                prev_stat = curr_stat;
                curr_stat = parseInt(strUser.substring(1)); [v.098] */
                var val = document.getElementById("X_" + curr_stat + "," + k);
                var strUser = val.options[val.selectedIndex].value;
                if (strUser == "NaN") {
                    maquina_encravada();
                    stopit = 1;
                } else {
                    prev_stat = curr_stat;
                    curr_stat = parseInt(strUser.substring(1));
                }
            }
        }
    }
    // Construção do grafico
    // http://www.graphviz.org/Gallery/directed/fsm.html
    // rankdir "TB", "LR", "BT", "RL", corresponding to directed graphs drawn from top to
bottom,
    // from left to right, from bottom to top, and from right to left, respectively.
    // Adicionamos um nó qi, para posteriormente criar o apontador para nó inicial
    cod_viz = "digraph finite_state_machine {\n rankdir=LR; qi";
    // para cada estado criamos um nó
    for (i = 0; i < nEstados; i++) cod_viz = cod_viz + " q" + i;
    cod_viz = cod_viz + ';\n size="8,5"\n';
    // atributo ponto qi branco para não ficar visivel
    cod_viz = cod_viz + " qi [shape=point color=white];\n";

    // atributos para os nos seguintes
    // o final fica com circulo duplo
    for (i = 0; i < nEstados; i++) {
```

```
    if (document.getElementById("Final" + i).checked) {
        // caso estejamos a correr simulação, então preenche de vermelho o nó final
        if (curr_stat == i && simulate == 1)
            cod_viz =
                cod_viz +
                " q" +
                i +
                " [shape=doublecircle style=filled color=black fillcolor=red];\n";
        else cod_viz = cod_viz + " q" + i + " [shape=doublecircle];\n";
    } else {
        // caso estejamos a correr simulação, então preenche de vermelho o nó não final
        if (curr_stat == i && simulate == 1)
            cod_viz =
                cod_viz +
                " q" +
                i +
                " [shape=circle style=filled color=black fillcolor=red];\n";
        else cod_viz = cod_viz + " q" + i + " [shape=circle];\n";
    }
}

// para cada estado ve as ligações para outros estados
var lig = [];
for (i = 0; i < nEstados; i++) {
    lig[i] = [];
    for (j = 0; j < nEstados; j++) {
        lig[i][j] = "";
    }
    for (j = 0; j < alfabeto.length; j++) {
        var val = document.getElementById("X_" + i + "," + j);
        var strUser = val.options[val.selectedIndex].value;
        to_state = parseInt(strUser.substring(1));
        lig[i][to_state] = lig[i][to_state] + alfabeto[j];
    }
}
```

```
    // apontador qi para nó inicial Exemplo: qi -> qo;
    cod_viz = cod_viz + " qi -> q" + estadoInicial + ";\n";
    // para cada um dos estados criar os nós de ligação com respectivo label
    // Exemplo: qo -> q1 [label="0,1" ];
    for (i = 0; i < nEstados; i++) {
        for (j = 0; j < nEstados; j++) {
            if (lig[i][j] != "") {
                //caso estejamos a correr uma simulação passo a passo, marca a vermelho a
passagem
                if (simulate == 1 && prev_stat == i && curr_stat == j)
                    cod_viz =
                        cod_viz +
                        " q" +
                        i +
                        " -> q" +
                        j +
                        ' [label="' +
                        lig[i][j].split("").join() +
                        '" color=red];\n';
                else
                    cod_viz =
                        cod_viz +
                        " q" +
                        i +
                        " -> q" +
                        j +
                        ' [label="' +
                        lig[i][j].split("").join() +
                        '" ];\n';
            }
        }
    }

    // fim da construção do gráfico
    cod_viz = cod_viz + "}\n";
```

```
    // TODO -> Ocultar quando não quiser que a consola imprima o
    // codigo disponiblizado para o Viz.JS
    // http://magjac.com/graphviz-visual-editor/
    // console.log(cod_viz);

    // (R)envia o grafico para o ecrã
    // O processo recoloca o grafico atualziado a cada passo simulado
    document.getElementById("graphviz_svg_div").innerHTML = "";
    var svgText = Viz(cod_viz, "svg");
    document.getElementById("graphviz_svg_div").innerHTML = "<hr>" + svgText;
    // Ativa div para input/output da simulação passo a passo
    document.getElementById("simula_viz").style.display = "block";

    //Se Simulação chegou ao fim, reinicia variaveis
    if (stopit == 1) {
        simulate = 0;
        pos_input = -2;
        prev_stat = -1;
        curr_stat = -1;
    }
}

// Função para mensagem maquina encravada add in [V.099]
function maquina_encravada() {
    if (lang == "EN") {
        document.getElementById("resultado").innerHTML =
            "No transition to the current symbol.<b style='color:red'> Jammed Machine.</b>";
    } else if (lang == "ES") {
        document.getElementById("resultado").innerHTML =
            "Sin transición al símbolo actual. <b style='color:red'>Máquina atascada.</b>";
    } else {
        document.getElementById("resultado").innerHTML =
            "Sem transição para o símbolo atual. <b style='color:red'>Máquina Encravada.</b>";
    }
}
```

```
// ao clicar botão de passo, para passo seguinte
function percorre() {
    // caso ainda não tenha sido iniciada a simulação
    if (simulate == 0) {
        pos_input = -2;
        prev_stat = -1;
        curr_stat = -1;
        // carrega input
        input = document.getElementById("input").value;
        // tamanho do input
        tam_input = input.length;
    }
    // ativa booleano para simulacao iniciada
    simulate = 1;
    // output para passo processado
    document.getElementById("resultado").innerHTML = "xxxxxxx";
    // Se chegamos ao fim, reiniciamos variaveis
    if (pos_input > tam_input) {
        pos_input = -2;
        prev_stat = -1;
        curr_stat = -1;
        document.getElementById("resultado").innerHTML = "";
    }
    // voltamos para a criação do grafico
    create_graph_desc();
}
// mensagem final multilingue (In)Sucesso
function messagem_final() {
    if (document.getElementById("Final" + curr_stat).checked) {
        if (lang == "EN") {
            document.getElementById("resultado").innerHTML = "Success!";
        } else if (lang == "ES") {
            document.getElementById("resultado").innerHTML = "Éxito!";
        } else {
```

```
            document.getElementById("resultado").innerHTML = "Successo!";
        }
    } else {
        if (lang == "EN") {
            document.getElementById("resultado").innerHTML = "Failed...";
        } else if (lang == "ES") {
            document.getElementById("resultado").innerHTML = "Fallido...";
        } else {
            document.getElementById("resultado").innerHTML = "Falhou...";
        }
    }
}

// mensagem de erro MultiLingue Simbolo nao existe
function mensagem_erro_sim(simb) {
    if (lang == "EN") {
        document.getElementById("resultado").innerHTML =
            "Unrecognized symbol: <b style='color:red'>" +
            simb +
            " </b>not in alphabet...";
    } else if (lang == "ES") {
        document.getElementById("resultado").innerHTML =
            "Símbolo no reconocido: <b style='color:red'>" +
            simb +
            " </b>no en el alfabeto...";
    } else {
        document.getElementById("resultado").innerHTML =
            "Símbolo não reconhecido: <b style='color:red'>" +
            simb +
            " </b>não está no alfabeto...";
    }
}

// Funcao para identicação do input consumido
function mostra_input_consumido(tipo_Sim, para_position, para_input) {
```

```
    var temp = "";
    for (i = 0; i < para_position; i++) {
        // coloca todo o input na variavel temporaria até à posição atual
        if (i < para_position) temp = temp + para_input.charAt(i);
    }
    // adiciona o marcador [ e o caracter consumido
    temp = temp + "[";
    if (para_position >= 0) {
        temp += para_input.charAt(i);
        i++;
    }
    // fecha o marcador
    temp += "]";
    // adiciona o restante input na variavel temporaria
    for (; i < para_input.length; i++) {
        temp = temp + para_input.charAt(i);
    }
    // Caso o pedido tenha vindo da tabela de simulação, retorna a variavel tempo
    // caso contrario, apresenta o resultado na respectiva div
    if (tipo_Sim == "charge") {
        return temp;
    } else {
        document.getElementById("inputConsumido").value = temp;
    }
}

// -- Tabela Simulações div "charge"
// Limpar conteudos da tabela de simulação
function limpar() {
    document.getElementById("run_simul").disabled = false;
    //-- Tratamento de Outputs Ativa/esconde Container--
    inOutDiv();
    document.getElementById("limp_simul").disabled = true;
}
```

```
// Função para chamar simulacao da base de testes e apresentar resultado na tabela
function simular() {
    // desativar o botao de correr base_tests
    document.getElementById("run_simul").disabled = true;
    // carrega as palavras a testar
    base_tests = document.getElementById("base_tests").value;
    // Inicializa variavel de entrada com as palavras pela quebra de linha
    var Inputs = base_tests.split("\n");
    // Incializa a mensagem de saida
    var Outputs = "";
    // para cada palavra corre a simulação
    for (var i = 0; i < Inputs.length; i++) {
        var para_input = Inputs[i];
        var res = simular_1(para_input);
        // para cada resultado é criada uma linha com 3 colunas
        Outputs =
            Outputs +
            // palavra em analise
            "<tr><td style='border: 1px solid black'>" +
            para_input +
            "</td><td style='border: 1px solid black'>" +
            // resultado
            res[0] +
            "</td></td><td style='border: 1px solid black'>" +
            // input consumido
            res[1] +
            "</td></tr>\n";
    }
    // cabeçalho da tabela
    Outputs =
        "<table id='simTable'>\n<tr><td bgcolor=lightgrey>input</td><td
bgcolor=lightgrey>resultado</td><td bgcolor=lightgrey>input consumido</td>\n" +
        Outputs +
        "</table>";
```

```
    // carrega resultados para a respetiva div
    document.getElementById("results").innerHTML = Outputs;
    // reativa o botão limpar simulação
    document.getElementById("limp_simul").disabled = false;
}

// Função de simulação de palavras
function simular_1(para_input) {
    var alfabeto = document.getElementById("symbols").value;
    var nEstados = parseInt(document.getElementById("nEstados").value);
    var stopit = 0;

    var primeiro = 0;
    var prev_stat = -1;
    var curr_stat = -1;
    var pos = -1;

    while (stopit == 0) {
        if (prev_stat == -1 && curr_stat == -1) {
            curr_stat = estadoInicial;
            primeiro = 1;
            continue;
        }
        pos++;

        if (pos == para_input.length) {
            if (document.getElementById("Final" + curr_stat).checked) {
                // fim com sucesso
                resultMG(1, curr_stat);
            } else {
                // fim sem sucesso
                resultMG(0, curr_stat);
            }

            stopit = 1;
```

```
        } else {
            simb = para_input[pos];
            for (var k = 0; k < alfabeto.length && alfabeto[k] != simb; k++);
            if (k == alfabeto.length) {
                // simbolo fora do alfabeto e paragem
                resultMG(2, curr_stat);
                stopit = 1;
            } else {
                var val = document.getElementById("X_" + curr_stat + "," + k);
                var strUser = val.options[val.selectedIndex].value;

                if (strUser == "NaN") {
                    maquina_encravada2(simb, curr_stat);
                    stopit = 1;
                } else {
                    prev_stat = curr_stat;
                    curr_stat = parseInt(strUser.substring(1));
                }
            }
        }
    }
    // variavel resultante em array com [resultado, input consumido]
    var resultado = [result, mostra_input_consumido("charge", pos, para_input)];
    return resultado;
}
// Funcao mensagem maquina encravada para tabela de simulação [v.099]
function maquina_encravada2(simb, curr_stat) {
    if (lang == "EN") {
        result =
            "No transition for the current symbol " +
            simb +
            " on state q" +
            curr_stat;
    } else if (lang == "ES") {
```

```
        result =
            "Sin transición para el actual símbolo " +
            simb +
            " en estado q" +
            curr_stat;
    } else {
        result =
            "Sem transição para o atual símbolo " + simb + " no estado q" + curr_stat;
    }

    return result;
}

// Mensagens MultiLingue das mensagens da simulação
function resultMG(info, curr_stat) {
    if (info == 1) {
        if (lang == "EN") {
            result =
                "<b style='color:green'> Success! </b>Reached final state q" +
                curr_stat;
        } else if (lang == "ES") {
            result =
                "<b style='color:green'>¡Éxito! </b>Estado final alcanzado q" +
                curr_stat;
        } else {
            result =
                "<b style='color:green'>Sucesso! </b>Estado final alcançado q" +
                curr_stat;
        }
    } else if (info == 0) {
        if (lang == "EN") {
            result =
                "<b style='color:red'>Failed.</b> Reached non final state q" +
                curr_stat;
        } else if (lang == "ES") {
```

```
            result =
                "<b style='color:red'>Fallido.</b> Estado no final alcanzado q" +
                curr_stat;
        } else {
            result =
                "<b style='color:red'>Falha.</b> Alcançou o estado não final q" +
                curr_stat;
        }
    } else {
        if (lang == "EN") {
            result =
                "Unrecognized symbol: <b style='color:red'> " +
                simb +
                "</b> not in alphabet...";
        } else if (lang == "ES") {
            result =
                "Símbolo no reconocido: <b style='color:red'>" +
                simb +
                "</b> no en alfabeto...";
        } else {
            result =
                "Símbolo não reconhecido: <b style='color:red'> " +
                simb +
                "</b> não está alfabeto...";
        }
    }
    return result;
}
// -- Fim! Tabela Simulações div "charge"

//-- Tratamento de Outputs Ativa/esconde Container--
// https://www.w3schools.com/jsref/prop_style_display.asp
function inOutDiv() {
    // ativa div tabela de testes
    document.getElementById("charge").style.display = "block";
```

```
    // carrega base de testes das variaveis globais e introduzidas
    document.getElementById("results").innerHTML =
        '<textarea id="base_tests" rows=15 cols=50>' + base_tests + "</textarea>";
    // ativa div com os resultados da tabela de testes
    document.getElementById("results").style.display = "block";
    // esconde ficheiro carregar, deixa fazer sentido nesta fase
    document.getElementById("file").style.display = "none";
}

// -- Processamento de Ficheiros
//-- Grava ficheiro para utilizar a posterior --
function grava_UAbALL() {
    // primeiro ler nestados
    nEstados = parseInt(document.getElementById("nEstados").value);
    // segundo captura simbolos
    alfabeto = document.getElementById("symbols").value;
    // inicia texto a guardar
    UAbALL_text = "";
    //adiciona ao texto, por esta ordem: nEstados + simbolos + estado inicial
    UAbALL_text =
        UAbALL_text + nEstados + "\n" + alfabeto + "\n" + estadoInicial + "\n";
    // marca estados finais com 1, os restantes com 0
    for (i = 0; i < nEstados; i++) {
        if (document.getElementById("Final" + i).checked) {
            UAbALL_text = UAbALL_text + "1";
        } else {
            UAbALL_text = UAbALL_text + "0";
        }
    }
    //adiciona quebra de linha
    UAbALL_text = UAbALL_text + "\n";
    // da quarta linha até a linha indexada pelo numero de estados
    // verifica transições para cada estado separando com ";",
    // para a sequencia e simbolos, pela ordem que foram carregados
    // adicionando uma quebra de linha no final de cada estado
```

```
    for (i = 0; i < nEstados; i++) {
        for (j = 0; j < alfabeto.length; j++) {
            var val = document.getElementById("X_" + i + "," + j);
            var strUser = val.options[val.selectedIndex].value;
            to_state = "";
            if (strUser != "") {
                to_state = parseInt(strUser.substring(1));
            }
            if (j > 0) UAbALL_text = UAbALL_text + ";";
            UAbALL_text = UAbALL_text + to_state;
        }
        UAbALL_text = UAbALL_text + "\n";
    }
    // adiciona base para casos de testes
    UAbALL_text = UAbALL_text + base_tests;

    // pede o nome do ficheiro multilingue
    if (lang == "EN") {
        fich = prompt("Insert filename (default='UAbALL.txt')", "UAbALL.txt");
    } else if (lang == "ES") {
        fich = prompt(
            "Insertar nombre de archivo (default='UAbALL.txt')",
            "UAbALL.txt"
        );
    } else {
        fich = prompt("Inserir Nome Ficheiro (default='UAbALL.txt')", "UAbALL.txt");
    }

    // processa gravação com nome selcionado, caso contrario com default
    if (fich != null) {
        if (fich == "") fich = "UAbALL.txt";
        // texto a salvar, nome do ficheiro
        saveTextAsFile(UAbALL_text, fich);
    }
}
```

```
// Download ficheiro como texto com recurso ao objecto Blob e URL.createObjectURL()
// https://developer.mozilla.org/pt-BR/docs/Web/API/Blob
// https://developer.mozilla.org/pt-BR/docs/Web/API/URL/createObjectURl
function saveTextAsFile(textToSave, fileNameToSaveAs) {
    var textoSalvar = new Blob([textToSave], { type: "text/plain" });
    var textToSaveAsURL = window.URL.createObjectURL(textoSalvar);
    var abc = document.createElement("a");
    var xyz = document.createTextNode("Tutorials");
    abc.append(xyz);
    abc.download = fileNameToSaveAs;
    abc.innerHTML = "Download File";
    abc.href = textToSaveAsURL;
    abc.style.display = "none";
    document.body.append(abc);
    abc.click();
    // alerta de sucesso - Metodo alert() - Com multilingue
    // https://www.w3schools.com/jsref/met_win_alert.asp
    if (lang == "EN") {
        alert("File stored in your download folder.");
    } else if (lang == "ES") {
        alert("Archivo guardado en la carpeta de descargas.");
    } else {
        alert("Ficheiro guardado na sua pasta de transferências.");
    }
}

//-- Carregamento de ficheiro --
function ler_UAbALL() {
    // Load file como texto para processamento com recurso ao objecto FileReader
    loadFileAsText(ler_UAbAllCont);
}
```

```
// Load file como texto para processamento com recurso ao objecto FileReader
// https://developer.mozilla.org/pt-BR/docs/Web/API/FileReader
function loadFileAsText(callBack) {
    var fileToLoad = document.getElementById("fileToLoad").files[0];

    var fileReader = new FileReader();
    fileReader.onload = function (fileLoadedEvent) {
        callBack(fileLoadedEvent.target.result);
    };
    fileReader.readAsText(fileToLoad, "UTF-8");
}
// -- Fim! Processamento de Ficheiros
```

[8]. CORPO.TXT

```
<div id="PT" style="text-align: center;">
<h1>Projecto de Engenharia Informática 2020 </h1>
<img src="./imgs/Logo.png" >
<h2>Automata Learning Lab</h2>
<img src="./imgs/software.png" style="width:40%;" >
</div>

<div id="ES" style="text-align: center;">
<h1>Proyecto de Ingeniería Informática 2020 </h1>
<img src="./imgs/Logo.png" >
<h2>Automata Learning Lab</h2>
<img src="./imgs/software.png" style="width:40%;" >
</div>

<div id="EN" style="text-align: center;">
<h1>Computer Engineering Project 2020  </h1>
<img src="./imgs/Logo.png" >
<h2>Automata Learning Lab</h2>
<img src="./imgs/software.png" style="width:40%;" >
</div>
```

[9]. DFA.TXT

```
<div id="PT">
    <p>
        <strong>DFA – Autómatos Finitos Deterministas</strong> – apresentam uma
        Função em que para cada estado e cada símbolo, possui uma única transição,
        começando em um estado inicial e avança para próxima célula após leitura de
        um símbolo. É representado formalmente por uma 5-tupla (Q, Σ, δ, qo, F),
        onde:
    </p>
    <ul>
        <li>Q é um conjunto finito de estados.</li>
        <li>Σ é um conjunto finito de símbolos, Alfabeto.</li>
        <li>δ é a função de transição, isto é, δ: Q x Σ → Q.</li>
        <li>
            qo é o estado inicial, isto é, o estado do autómato antes de qualquer
            entrada ser processada, onde qo ∈ Q
        </li>
        <li>
            F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de
            aceitação.
        </li>
    </ul>
    <hr />
    <!--  Simulador  PT-->
    <h3>SIMULADOR</h3>
    <!-- Tabela para carregamento dados -->
    <table>
        <tr>
            <td id="simula_carga">
                <div>
                    <!-- Carregamento de ficheiro -->
                    <div id="file">
                        <label for="fileToLoad"
                            >Carregar um ficheiro UAbALL guardado:</label>
```

```
    <input
        type="file"
        id="fileToLoad"
        name="fileToLoad"
        onchange="ler_UAbALL()"
    />
    <br /><br />
</div>
<!-- Carregamento numero de estados -->
<label for="nEstados">Número de Estados: </label>
<input
    type="number"
    id="nEstados"
    min="2"
    max="30"
    size="3"
    value="2"
    name="nEstados"
/>
<br /><br />
<!-- Carregamento simbolos -->
<label for="symbols"
    >Alfabeto, conjunto finito de símbolos. &Sigma; =</label
>
<input
    type="Symbols"
    id="symbols"
    size="9"
    value="01"
    name="symbols"
/>
<br /><em
    >(colocar os símbolos seguidos, como no exemplo, sem qualquer
    separação)</em>
<br /><br />
```

```
                <!-- Ativar tabela transições -->
                <label for="trasitions">Ativar tabela transições : </label>

                <input
                    type="button"
                    id="constroi_transitions"
                    name="trasitions"
                    value="Ativar &delta;"
                    onclick="constroi_transitions()"
                />
                <br /><br />
                <!-- espaço para tabela de transições gerada em dfa.js-->
                <table id="transitions"></table>

            </div>
        </td>
        <td>
            <!-- coluna que recebe tabela de testes a simular-->
            <div id="charge" style="display: none;">
                <button id="run_simul" onclick="simular()">Simular</button>
                <button id="limp_simul" onclick="limpar()" disabled="true">
                    Limpar
                </button>
                <br />
                <div id="results"></div>
            </div>
        </td>
    </tr>
</table>
<!-- Target for dynamic svg generation -->
<div id="graphviz_svg_div"></div>
<!-- Tabela para interação com simulador viz -->
<table id="passos">
    <tr>
```

```
            <td>
                <div id="simula_viz" style="display: none;">
                <h4> Simulação passo-a-passo</h4>
                <!-- Simulação passo-a-passo -->
                <label for="trasitions">Inserir input: </label>
                <input id="input" size="35" name="trasitions"/>
                <br /><em
                        >(colocar os símbolos seguidos sem qualquer separação)</em>
                <br /><br />
                <label for="inputConsumido">input consumido: </label>
                <input
                    id="inputConsumido"
                    size="30"
                    readonly="readonly"
                />
                <br /><br />
                <button id="simula_passo" onclick="percorre()">Passo</button>
                <em id="resultado"></em>
            </td>
        </tr>
    </table>
</div>

<!-- ++++++++++++++++++ END PT ++++++++++++++++++++++ -->

<div id="ES">
    <p>
        <strong>Los DFA - Autómatas Determinantes Finitos</strong> - presentan una
        Función en la que para cada estado y cada símbolo, tiene una única
        transición, comenzando en un estado inicial y pasando a la siguiente celda
        después de leer un símbolo. Está formalmente representado por una 5-tupla
        (Q, Σ, δ, qo, F), donde:
    </p>
    <ul>
        <li>Q es un conjunto finito de estados.</li>
```

```
    <li>Σ es un conjunto finito de símbolos, el Alfabeto.</li>
    <li>δ es la función de transición, es decir δ: Q x Σ → Q.</li>
    <li>
        qo es el estado inicial, es decir, el estado del autómata antes de que se
        procese cualquier entrada, donde qo ∈ Q.
    </li>
    <li>
        F es un conjunto de estados Q (es decir, F ⊆ Q) llamados estados de
        aceptación.
    </li>
</ul>
<hr />
<!--  Simulador  ES-->
<h3>SIMULADOR</h3>
<!-- Tabla para cargar datos -->
<table>
    <tr>
        <td id="simula_carga">
            <div>
                <!-- Subir archivo -->
                <div id="file">
                    <label for="fileToLoad"
                        >Cargar un archivo guardado UAbALL:</label>
                    <input
                        type="file"
                        id="fileToLoad"
                        name="fileToLoad"
                        onchange="ler_UAbALL()"
                    />
                    <br /><br />
                </div>
                <!-- Cargando número de estados -->
                <label for="nEstados">Número de Estados: </label>
                <input
```

```
    type="number"
    id="nEstados"
    min="2"
    max="30"
    size="3"
    value="2"
    name="nEstados"
/>
<br /><br />
<!-- Cargando símbolos -->
<label for="symbols"
    >Alfabeto, conjunto finito de símbolos. &Sigma; =</label
>
<input
    type="Symbols"
    id="symbols"
    size="9"
    value="01"
    name="symbols"
/>
<br /><em
    >(colocar los símbolos en una fila, como en el ejemplo, sin ninguna
    separación)</em                    >
<br /><br />
<!--Habilitar transiciones de tabla -->
<label for="trasitions">Habilitar transiciones de tabla : </label>
<input
    type="button"
    id="constroi_transitions"
    name="trasitions"
    value="Activar &delta;"
    onclick="constroi_transitions()"
/>
<br /><br />
<!-- espacio para tabla de transición genera en dfa.js-->
```

```
                <table id="transitions"></table>
            </div>
        </td>
        <td>
            <!-- coluna que recebe tabela de testes a simular-->
            <div id="charge" style="display: none;">
                <button id="run_simul" onclick="simular()">Simular</button>
                <button id="limp_simul" onclick="limpar()" disabled="true">
                    Borrar
                </button>
                <br />
                <div id="results"></div>
            </div>
        </td>
    </tr>
</table>
<!-- Target for dynamic svg generation -->
<div id="graphviz_svg_div"></div>
<!-- Tabla para la interacción con el simulador viz -->
<table id="passos">
    <tr>
        <td>
            <div id="simula_viz" style="display: none;">
            <h4> Paso a paso de simulación</h4>
            <!-- Paso a paso de simulación -->
            <label for="trasitions">Poner input: </label>
            <input id="input" size="10" name="trasitions"/>
            <br /><em
                    >(colocar los símbolos en una fila sin ninguna separación)</em>
            <br /><br />
            <label for="inputConsumido">input consumido: </label>
            <input
                id="inputConsumido"
                size="10"
```

```
                readonly="readonly"
            />
            <br /><br />
            <button id="simula_passo" onclick="percorre()">Paso</button>
            <em id="resultado"></em>
        </td>
    </tr>
</table>



</div>

<!-- ++++++++++++++++++++++++++ END ES ++++++++++++++++++++++++++ -->
<div id="EN">
    <p>
        <strong>DFA - Deterministic Finite Automata </strong>- presents a Function
        in which, for every state and each symbol, there is a single transition,
        starting in an initial state and progressing to the next cell after reading
        a symbol. It is formally represented by a 5-tuple (Q, Σ, δ, qo, F), where:
    </p>
    <ul>
        <li>Q is a finite set of states.</li>
        <li>Σ is a finite set of symbols, Alphabet.</li>
        <li>δ is the transition function, that is, δ: Q x Σ → Q.</li>
        <li>
            qo is the initial state, that is, the state of the automata before any
            input is processed, where qo ∈ Q
        </li>
        <li>
            F is a set of states of Q (that is, F ⊆ Q) called acceptance states.
        </li>
    </ul>
    <hr />
    <!--  Simulator  EN-->
```

```
<h3>SIMULADOR</h3>
<!-- Table for given loading -->
<table>
    <tr>
        <td id="simula_carga">
            <div>
                <!-- File upload -->
                <div id="file">
                    <label for="fileToLoad"
                        >Load a saved file UAbALL:</label
                    >
                    <input
                        type="file"
                        id="fileToLoad"
                        name="fileToLoad"
                        onchange="ler_UAbALL()"
                    />
                    <br /><br />
                </div>
                <!-- Charging number of states -->
                <label for="nEstados">Number of States: </label>
                <input
                    type="number"
                    id="nEstados"
                    min="2"
                    max="30"
                    size="3"
                    value="2"
                    name="nEstados"
                />
                <br /><br />
                <!-- Loading symbols -->
                <label for="symbols"
                    >Alphabet, finite set of symbols. &Sigma; =</label>
```

```
            <input
                type="Symbols"
                id="symbols"
                size="9"
                value="01"
                name="symbols"
            />
            <br /><em
                >(put the row symbols, as in the example, without any separation)</em
            >
            <br /><br />
            <!-- Ativar tabela transições -->
            <label for="trasitions">Enable table transitions : </label>
            <input
                type="button"
                id="constroi_transitions"
                name="trasitions"
                value="Enable &delta;"
                onclick="constroi_transitions()"
            />
            <br /><br />
            <!-- transitions table space generated in dfa.js-->
            <table id="transitions"></table>
        </div>
    </td>
    <td>
        <!-- coluna que recebe tabela de testes a simular-->
        <div id="charge" style="display: none;">
            <button id="run_simul" onclick="simular()">Simulate</button>
            <button id="limp_simul" onclick="limpar()" disabled="true">
                Clean
            </button>
            <br />
            <div id="results"></div>
        </div>
```

```
            </td>
        </tr>
    </table>
    <!-- Target for dynamic svg generation -->
    <div id="graphviz_svg_div"></div>
    <!-- Table for interaction with simulator viz -->
    <table id="passos">
        <tr>
            <td>
                <div id="simula_viz" style="display: none;">
                <h4> Step-by-step simulation</h4>
                <!-- Step-by-step simulation -->
                <label for="trasitions">Insert input: </label>
                <input id="input" size="10" name="trasitions"/>
                <br /><em
                        >(put the row symbols without any separation)</em>
                <br /><br />
                <label for="inputConsumido">consumed input: </label>
                <input
                    id="inputConsumido"
                    size="10"
                    readonly="readonly"
                />
                <br /><br />
                <button id="simula_passo" onclick="percorre()">Step</button>
                <em id="resultado"></em>
            </td>
        </tr>
    </table>
</div>

<!-- ++++++++++++++++++++++++ END EN +++++++++++++++++ -->
```

[10]. ENFA.TXT

```
<div id="PT">
<p><strong>NFA-</strong><strong>ε – Autómatos Finitos Não-Deterministas com Movimentos Vazios </strong>– semelhantes aos NFA, acrescendo a possibilidade de transições de estados sem leitura de um símbolo da Fita. É representado formalmente por uma 5-tupla (Q, Σ, δ, qo, F), onde: </p>
<ul>
    <li>Q é um conjunto finito de estados.</li>
    <li>Σ é um conjunto finito de símbolos, Alfabeto.</li>
    <li>δ é a função de transição, isto é, δ: Q x Σ → P(Q).
<ul>
    <li>P(Q) conjunto das partes de Q.</li>
</ul>
</li>
    <li>qo é o estado inicial, onde qo ∈ Q.</li>
    <li>F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.</li>
</ul>
<hr/>
</div>


<div id="ES">
<p><strong>NFA-ε - Autómatas finitos no deterministas con movimientos vacíos </strong>- similar a NFA, añadiendo la posibilidad de transiciones de estado sin leer un símbolo de la cinta. Está formalmente representado por una 5-tupla (Q, Σ, δ, qo, F), donde:</p>
<ul>
    <li>Q es un conjunto finito de estados.</li>
    <li>Σ es un conjunto finito de símbolos, el Alfabeto.</li>
    <li>δ es la función de transición, es decir δ: Q x Σ → P(Q).</li>
    <li>P(Q) conjunto de partes de Q.</li>
    <li>qo es el estado inicial, donde qo ∈ Q.</li>
    <li>F es un conjunto de estados de Q (es decir, F ⊆ Q) llamados estados de aceptación.</li>
</ul>
<hr/>
</div>
```

```
<div id="EN">
<p><strong>NFA - Finite Non-Deterministic Automata whith Empty Movements </strong>- similar to NFA, adding the possibility of state transitions without reading a Ribbon symbol. It is formally represented by a 5-tuple (Q, Σ, δ, q0, F), where:</p>
<ul>
    <li>Q is a finite set of states.</li>
    <li>Σ is a finite set of symbols, Alphabet.</li>
    <li>δ is the transition function, that is, δ: Q x Σ → P (Q).</li>
    <li>P (Q) set of parts of Q.</li>
    <li>q0 is the initial state, where q0 ∈ Q.</li>
    <li>F is a set of states of Q (that is, F ⊆ Q) called acceptance states.</li>
</ul>
<hr/>
</div>
```

[11]. FOOTER.TXT

```
<p id="PT">Projecto de Engenharia Informática 2020 de <a href="https://www.andremaciel.pt">André Maciel Sousa</a>
<small>V.1.0</small> </p>
<p id="EN">Computer Engineering Project 2020 of <a href="https://www.andremaciel.pt">André Maciel Sousa</a>
<small>V.1.0</small> </p>
<p id="ES">Proyecto de Ingeniería Informática 2020 de <a href="https://www.andremaciel.pt">André Maciel Sousa</a>
<small>V.1.0</small> </p>
```

[12]. GRAMMAR.TXT

```
<div id="PT">


<p><strong>GRAMMAR – Gramática Livre de Contexto -</strong> são modelos uteis na conceção de software que processa estrutura de dados recursivamente. O exemplo mais conhecido é o "<em>parsing</em>", componente de um compilador que lida com os recursos aninhados recursivamente da linguagem de programação, como expressões aritméticas condicionais. É representada formalmente por uma 4-upla (V, T, P, S), onde: </p>
<ul>
```

```
    <li>V é o conjunto de variáveis</li>
    <li>T é o conjunto de terminais</li>
    <li>P é o conjunto de produções</li>
    <li>S o símbolo inicial, que deverá ser elemento de V.</li>
</ul>
<p>Um exemplo de uma Gramática Livre de Contexto para números binários compostos por n zeros seguidos de n uns é dado por CFG=({S}, [0,1], {S-&gt;01, S-&gt;0S1}, {S}). </p


<p>Existem algumas operações possíveis de realizar sobre autómatos, as quais deverão ser alvo de versões posteriores deste Laboratório, como a conversão NFA-ε para NFA, NFA para DFA, DFA para Expressões Regulares, Expressões Regulares para NFA-ε e Gramática Livre de Contexto para PDA. </p>
<hr/>
</div>


<div id="ES">


<p><strong>GRAMÁTICA - Gramática Libre de Contexto - </strong>son modelos útiles en el diseño de software que procesa la estructura de los datos de forma recursiva. El ejemplo más conocido es el "análisis sintáctico", un componente del compilador que se ocupa de las características anidadas recursivamente del lenguaje de programación, como las expresiones aritméticas condicionales. Está formalmente representado por un 4-tupla (V, T, P, S), donde:</p>
<ul>
    <li>V es el conjunto de variables</li>
    <li>T es el conjunto de terminales</li>
    <li>P es el conjunto de producciones</li>
    <li>S el símbolo inicial, que debe ser elemento de V.</li>
</ul>
<p>Un ejemplo de una Gramática Libre de Contexto para números binarios compuestos de números seguidos de n es dado por CFG=({S}, [0.1}, {S-&gt;01, S-&gt;0S1}, {S}).</p>


<p>Hay algunas posibles operaciones a realizar en los autómatas, que deberían ser objeto de versiones posteriores de este Laboratorio, como la conversión de NFA-ε a NFA, NFA a DFA, DFA a Expresiones Regulares, Expresiones Regulares a NFA-ε y Gramática Libre de Contexto para PDA.</p>
<hr/>
</div>
```

```
<div id="EN">


<p><strong>GRAMMAR - Context-Free Grammar</strong> - are useful models in the design of software that processes data structure recursively. The best-known example is “parsing”, a compiler component that deals with the recursively nested resources of the programming language, such as conditional arithmetic expressions. It`s formally represented by a 4-upla (V, T, P, S), where: </p>

<ul>

    <li>V is the set of variables</li>

    <li>T is the set of terminals</li>

    <li>P is the set of productions</li>

    <li>S the initial symbol, which should be a V element.</li>

</ul>

<p>An example of a Context-Free Grammar for binary numbers composed of n zeros followed by n ones is given by CFG = ({S}, [0,1}, {S-&gt; 01, S-&gt; 0S1}, {S}).</p>

<p>There are some viable operations to be carried out on automata, which should be the target of later versions of this Laboratory, such as converting NFA-ε to NFA, NFA to DFA, DFA for Regular Expressions, Regular Expressions for NFA-ε and Context-Free Grammar for PDA.</p>

<hr/>

</div>
```

[13]. MENU.TXT

```
<ul id="menu">
<li onclick="main(home);"><a href="#home">Home</a></li>
<li onclick="main(dfa);"><a href="#dfa">DFA</a></li>
<li onclick="main(nfa);"><a href="#nfa">NFA</a></li>
<li onclick="main(enfa);"><a href="#nfae">NFA-ε</a></li>
<li onclick="main(pda);"><a href="#pda">PDA</a></li>
<li onclick="main(turing);"><a href="#turing">TURING</a></li>
<li onclick="main(regex);"><a href="#regex">REGEX</a></li>
<li onclick="main(grammar);"><a href="#grammar">GRAMMAR</a></li>
<li class="esp"><a href="./start.html?lang=ES"></a></li>
<li class="eng"><a href="./start.html?lang=EN"></a></li>
<li class="prt"><a href="./start.html?lang=PT"></a></li>
</ul>
```

[14]. NFA.TXT

```
<div id="PT">
<p><strong>NFA – Autómatos Finitos Não-Deterministas</strong> – semelhantes aos DFA, diferenciando-se na Função, que ao processar uma entrada composta pelo estado corrente e o símbolo lido, tem como resultado um conjunto de novos estados. É representado formalmente por uma 5-tupla (Q, Σ, δ, qo, F), onde:</p>
<ul>
    <li>Q é um conjunto finito de estados.</li>
    <li>Σ é um conjunto finito de símbolos, Alfabeto.</li>
    <li>δ é a função de transição, isto é, δ: Q x Σ → P(Q).
<ul>
    <li>P(Q) conjunto das partes de Q.</li>
</ul>
</li>
    <li>qo é o estado inicial, onde qo ∈ Q.</li>
    <li>F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.</li>
</ul>
<hr/>
</div>
<div id="ES">


<p><strong>NFA - Autómatas finitos no deterministas - </strong>similares al DFA, diferenciándose en la Función, que al procesar una entrada compuesta por el estado actual y el símbolo leído, resulta en un conjunto de nuevos estados. Está formalmente representado por una 5-tupla (Q, Σ, δ, qo, F), donde:</p>
<ul>
    <li>Q es un conjunto finito de estados.</li>
    <li>Σ es un conjunto finito de símbolos, el Alfabeto.</li>
    <li>δ es la función de transición, es decir δ: Q x Σ → P(Q).</li>
    <li>P(Q) conjunto de partes de Q.</li>
    <li>qo es el estado inicial, donde qo ∈ Q.</li>
    <li>F es un conjunto de estados de Q (es decir, F ⊆ Q) llamados estados de aceptación.</li>
</ul>
<hr/>
</div>
```

```
<div id="EN">
<p><strong>NFA - Finite Non-Deterministic Automata </strong>- similar to DFA, differing in Function, which, when processing an entry composed of the current state and the read symbol, results in a set of new states. It is formally represented by a 5-tuple (Q, Σ, δ, q0, F), where:</p>
<ul>
    <li>Q is a finite set of states.</li>
    <li>Σ is a finite set of symbols, Alphabet.</li>
    <li>δ is the transition function, that is, δ: Q x Σ → P (Q).</li>
    <li>P (Q) set of parts of Q.</li>
    <li>q0 is the initial state, where q0 ∈ Q.</li>
    <li>F is a set of states of Q (that is, F ⊆ Q) called acceptance states.</li>
</ul>
<hr/>
</div>
```

[15]. PDA.TXT

```
<div id="PT">


<p><strong>PDA – Autómato Pilha</strong> – também conhecidos como autómatos <em>push-down </em>são máquinas semelhantes aos autómatos finitos anteriores, acoplado a uma pilha que funciona como memoria auxiliar, que pode armazenar uma <em>String </em>de comprimento arbitrário. A pilha pode ser lida e modificada apenas na parte superior. É representado formalmente por uma 6-tupla (Q, Σ, Γ, Δ, q0, F), onde:</p>
<ul>
    <li>Q é um conjunto finito de estados.</li>
    <li>Σ é um conjunto finito de símbolos, Alfabeto.</li>
    <li>Γ é um conjunto finito de símbolos, Alfabeto da pilha.</li>
    <li>Δ ⊆ (Q X Σ* X Γ*) X (Q × Γ*) é a relação de transição.</li>
    <li>q0 é o estado inicial, onde q0 ∈ Q.</li>
    <li>F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.</li>
</ul>
<hr/>
</div>
```

```
<div id="ES">


<p><strong>PDA - Autómata de pila</strong> - también conocido como autómata de empuje (push-down), son máquinas similares a los autómatas finitos anteriores, acopladas a una pila que funciona como una memoria auxiliar, que puede almacenar una cadena de longitud arbitraria. La pila sólo puede ser leída y modificada en la parte superior. Está formalmente representado por un 6-tupla (Q, Σ, Γ, Δ, qo, F), donde:</p>

<ul>

    <li>Q es un conjunto finito de estados.</li>

    <li>Σ es un conjunto finito de símbolos, el Alfabeto.</li>

    <li>Γ es un conjunto finito de símbolos, Alfabeto de la pila.</li>

    <li>Δ ⊆ Δ ⊆ (Q X Σ* X Γ*) X (Q × Γ*) es el ratio de transición.</li>

    <li>qo es el estado inicial, donde qo ∈ Q.</li>

    <li>F es un conjunto de estados Q (es decir, F ⊆ Q) llamados estados de aceptación.</li>

</ul>

<hr/>

</div>


<div id="EN">

<p><strong>PDA - Stack Automaton</strong> - also known as push-down automata are machines similar to former finite automata, coupled to a stack that functions as an auxiliary memory, which can store a string of arbitrary length. The stack can only be read and modified at the top. It is formally represented by a 6-tuple (Q, Σ, Γ, Δ, qo, F), where: </p>

<ul>

    <li>Q is a finite set of states.</li>

    <li>Σ is a finite set of symbols, Alphabet.</li>

    <li>Γ is a finite set of symbols, the Stack alphabet.</li>

    <li>Δ ⊆ (Q X Σ* X Γ*) X (Q × Γ*) is the transition relationship.</li>

    <li>qo is the initial state, where qo ∈ Q.</li>

    <li>F is a set of states of Q (that is, F ⊆ Q) called acceptance states.</li>

</ul>

<hr/>

</div>
```

[16]. REGEX.TXT

```
<div id="PT">

<p>As linguagens regulares constituem a classe de linguagens com menor poder de representação, sendo possível desenvolver algoritmos de reconhecimento, existindo várias aplicações, como a analise léxica, sistemas de animação, hipertextos e hipermédia (Menezes, 2000). As linguagens regulares podem ser presentadas por um autómato finito e por uma Expressão Regular (REGEX).</p>

<p><strong>REGEX – Expressões Regulares</strong> – uma forma sequencial de especificar uma linguagem regular, através de um padrão de <em>Strings </em>que descreve o mesmo que pode ser descrito por um autómato finito.  Um exemplo, em notação UNIX de uma REGEX " [A-Z][a-z]*[ ][A-Z][A-Z]"  representa uma palavra iniciada com maiúscula, seguida de espaço e duas maiúsculas, nesta seria aceite a sequência "Porto PT".</p>

<hr/>

</div>

<div id="ES">

<p>Los lenguajes regulares constituyen la clase de lenguajes con menor poder de representación, siendo posible desarrollar algoritmos de reconocimiento, existiendo varias aplicaciones como el análisis léxico, los sistemas de animación, los hipertextos y los hipermedios (Menezes, 2000). Los lenguajes regulares pueden ser presentados por un autómata finito y por una Expresión Regular (REGEX).</p>


<p><strong>REGEX - Expresiones Regulares - </strong>una forma secuencial de especificar un lenguaje regular, a través de un patrón de cadenas que describe lo mismo que puede ser descrito por un autómata finito.  Un ejemplo, en la notación UNIX de un REGEX "A-Za-z*[ ][A-Z][A-Z]" representa una palabra que comienza con una letra mayúscula, seguida de un espacio y dos letras mayúsculas, en la que se aceptaría la secuencia "Puerto PT".</p>

<hr/>

</div>

<div id="EN">

<p>Regular languages constitute the class of languages with less power of representation, and it is possible to develop recognition algorithms, with several applications, such as lexical analysis, animation systems, hypertexts, and hypermedia (Menezes, 2000). Regular languages can be presented by a finite automaton and a Regular Expression (REGEX). </p>


<p><strong>REGEX - Regular Expressions</strong> - a sequential way to specify a regular language, through a pattern of Strings that explains the same that can be described by a finite automaton. An example, in UNIX notation of a REGEX "[A-Z] [a-z] * [] [A-Z] [A-Z]" represents a word beginning with a capital letter, followed by a space and two capital letters, in which the sequence "Porto PT" would be accepted. </p>

<hr/>

</div>
```

[17]. TURING.TXT

```
<div id="PT">


<p><strong>TURING – Máquina de Turing</strong> – os autómatos desta classe são capazes de executar quaisquer tarefas que sejam efetivamente computáveis.  Na MT a Fita é extensível para a esquerda e para a direita, i.e.., a MT possui a Fita necessária à computação onde as células ainda não preenchidas estão preenchidas com um símbolo especial <em>branco</em>. A Unidade de Controlo (cabeça) lê e escreve símbolos na fita movendo-se para esquerda ou direita. A Função de Transição além do movimento indica que símbolo deverá ser escrito, quando não existirem entrada na tabela para a combinação atual de símbolo e estado a máquina para. É representada formalmente por uma 7-upla (Q, Σ, Γ, qo, b, F, δ), onde:</p>

<ul>

    <li>Q é um conjunto finito de estados.</li>

    <li>Σ é um conjunto finito de símbolos, Alfabeto.</li>

    <li>Γ é um conjunto finito de símbolos, Alfabeto da Fita.</li>

    <li>qo é o estado inicial, onde qo ∈ Q.</li>

    <li>b é o símbolo branco, onde b ∈ Γ.</li>

    <li>F é um subconjunto de estados de Q (isto é, F ⊆ Q) chamado de estados de aceitação.</li>

    <li>δ: é a relação de transição, em que R e L são as direções direita e esquerda, respetivamente.</li>

</ul>

<hr/>

</div>


<div id="ES">

<p><strong>TURING – Máquina de Turing (MT)</strong> –Los autómatas de esta clase son capaces de realizar cualquier tarea que sea efectivamente computable.  En la MT la cinta es extensible a la izquierda y a la derecha, es decir, la MT tiene la cinta de cálculo necesaria donde los nodos que aún no están llenos se completan con un símbolo especial blanco. La unidad de control (cabeza) lee y escribe símbolos en la cinta moviéndose a la izquierda o a la derecha. La función de transición más allá del movimiento indica qué símbolo debe escribirse cuando no hay ninguna entrada en la tabla para la combinación actual de símbolo y el estado en que la máquina se detiene. Está formalmente representado por un 7-tupla (Q, Σ, Γ, qo, b, F, δ), donde: </p>

<ul>

    <li>Q es un conjunto finito de estados.</li>

    <li>Σ es un conjunto finito de símbolos, el Alfabeto.</li>
```

Γ es un conjunto finito de símbolos, el Alfabeto de la Cinta.

qo es el estado inicial, donde qo ∈ Q.

b es el símbolo blanco, donde b ∈ Γ.

F es un conjunto de estados de Q (es decir, F ⊆ Q) llamados estados de aceptación.

δ: Q×⇒Q × Γ ×{L,R} es el ratio de transición, donde R y L son las direcciones derecha e izquierda respectivamente.

<hr/>

TURING - Turing machine - the automata of this class are capable of performing any effectively computable tasks. In the MT the Tape is extendable to the left and the right, i.e., the MT has the Tape necessary for computing where the cells not yet filled are filled with a special white symbol. The Control Unit (head) reads and writes symbols on the tape moving left or right. The Transition Function in addition to the movement indicates which symbol should be written, when there is no entry in the table for the current combination of symbol, and state the machine stops. It is formally represented by a 7-upla (Q, Σ, Γ, qo, b, F, δ), where:

Q is a finite set of states.

Σ is a finite set of symbols, Alphabet.

Γ is a finite set of symbols, Alphabet of tape.

qo is the initial state, where qo ∈ Q.

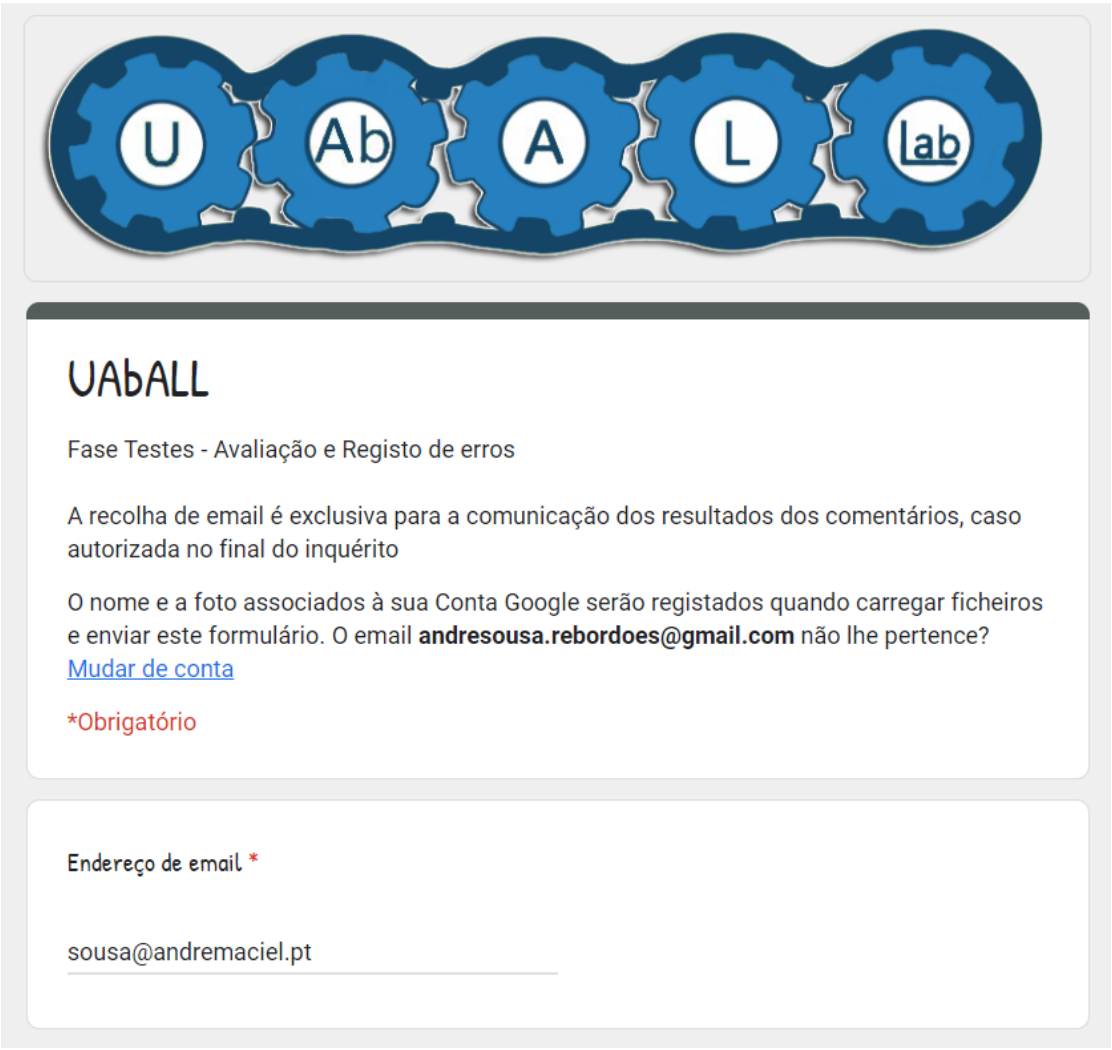
b is the white symbol, where b ∈ Γ.

F is a set of states of Q (that is, F ⊆ Q) called acceptance states.

δ: Q × Γ⇒Q × Γ × {L, R} is the transition relationship, where R and L are the right and left directions, respectively.

<hr/>

[18]. INQUÉRITO



UAbALL

Fase Testes - Avaliação e Registo de erros

A recolha de email é exclusiva para a comunicação dos resultados dos comentários, caso autorizada no final do inquérito

O nome e a foto associados à sua Conta Google serão registados quando carregar ficheiros e enviar este formulário. O email **andresousa.rebordoes@gmail.com** não lhe pertence? Mudar de conta

*Obrigatório

Endereço de email *

sousa@andremaciel.pt

Curso *

- ○ Licenciatura em Ciências do Ambiente
- ○ Licenciatura em Ciências Sociais
- ○ Licenciatura em Educação
- ◉ Licenciatura em Engenharia Informática
- ○ Licenciatura em Estudos Artísticos
- ○ Licenciatura em Estudos Europeus
- ○ Licenciatura em Gestão
- ○ Licenciatura em História
- ○ Licenciatura em Humanidades
- ○ Licenciatura em Línguas Aplicadas
- ○ Licenciatura em Matemática Aplicada à Gestão
- ○ Licenciatura em Matemática e Aplicações

Seguinte

Página 1 de 8

Licenciatura em Engenharia Informática

UC 21078- LINGUAGENS E COMPUTAçãO *

Se já frequentou esta unidade curricular coloque SIM

- ◯ Sim
- ◯ Não

Laboratórios Integrados no Moodle *

Considere importante a existência de laboratórios práticos integrados na plataforma?

- ◯ Sim
- ◯ Não

UABALL *

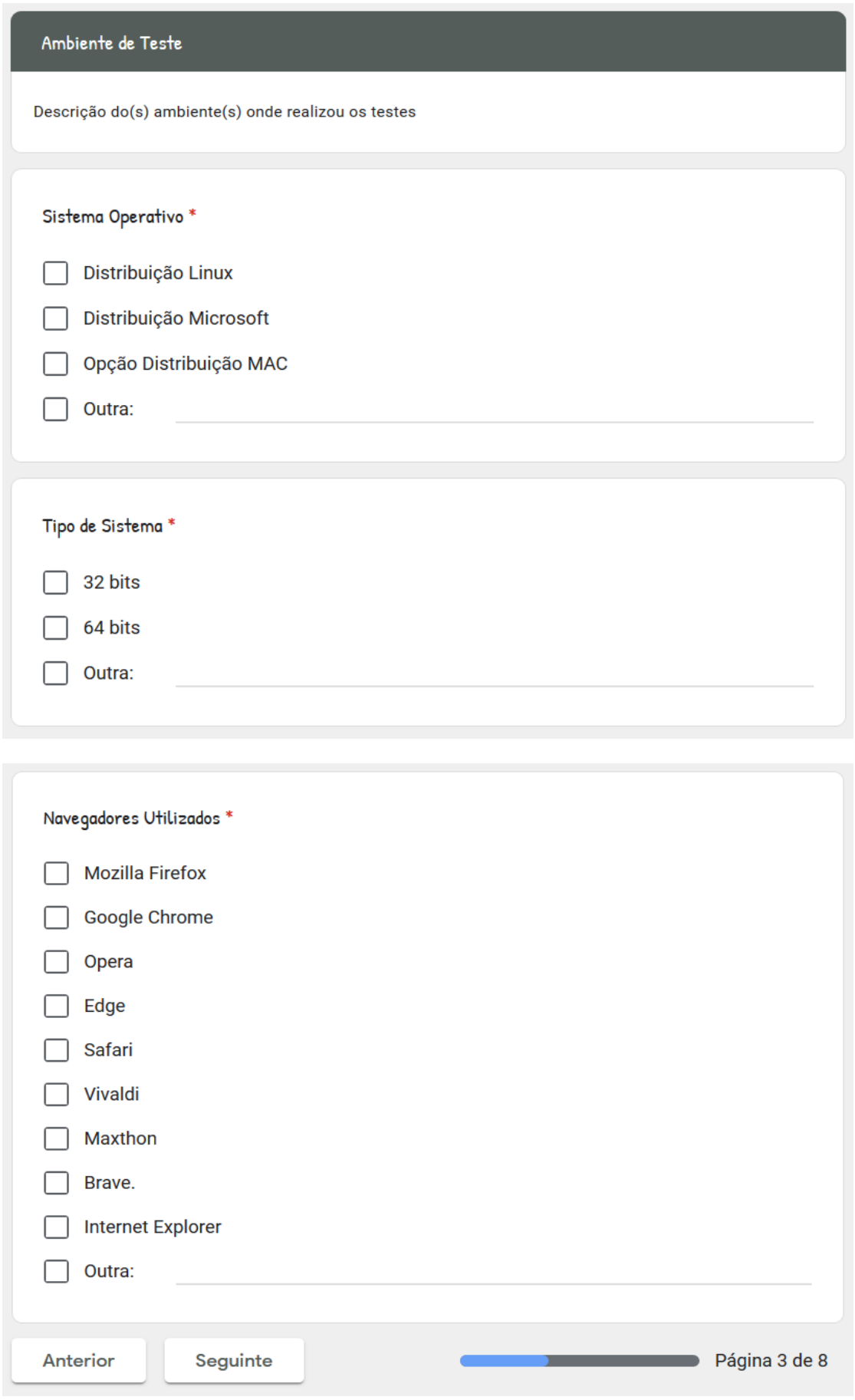
Considera que este projecto poderá ser uma mais valia para a UC 21078- LINGUAGENS E COMPUTAçãO

- ◯ Sim
- ◯ Não

Consideraria Participar na produção de um dos módulos restantes? *

- ◯ Sim
- ◯ Não
- ◯ Talvez

Anterior Seguinte Página 2 de 8



Ambiente de Teste

Descrição do(s) ambiente(s) onde realizou os testes

Sistema Operativo *

- [ ] Distribuição Linux
- [ ] Distribuição Microsoft
- [ ] Opção Distribuição MAC
- [ ] Outra: _____

Tipo de Sistema *

- [ ] 32 bits
- [ ] 64 bits
- [ ] Outra: _____

Navegadores Utilizados *

- [ ] Mozilla Firefox
- [ ] Google Chrome
- [ ] Opera
- [ ] Edge
- [ ] Safari
- [ ] Vivaldi
- [ ] Maxthon
- [ ] Brave.
- [ ] Internet Explorer
- [ ] Outra: _____

Anterior Seguinte Página 3 de 8

opções utilizadas

Nesta versão o laboratório tem opções como Multilingue e DFA disponíveis, sendo que neste permite:
Construir manualmente com identificação de numero de estados, símbolos e tabela de transição
Salvar o autómato criado
Carregar um autómato previamente salvo
Construir o grafo
Simular passo a passo com visualização no grafo
Colocar palavras a simular numa tabela e simular

Utilizou opção Multilingue? *

- ◯ Sim
- ◯ Não

Utilizou a construção manual do DFA? *

- ◯ Sim
- ◯ Não

utilizou Guardar Autómato? *

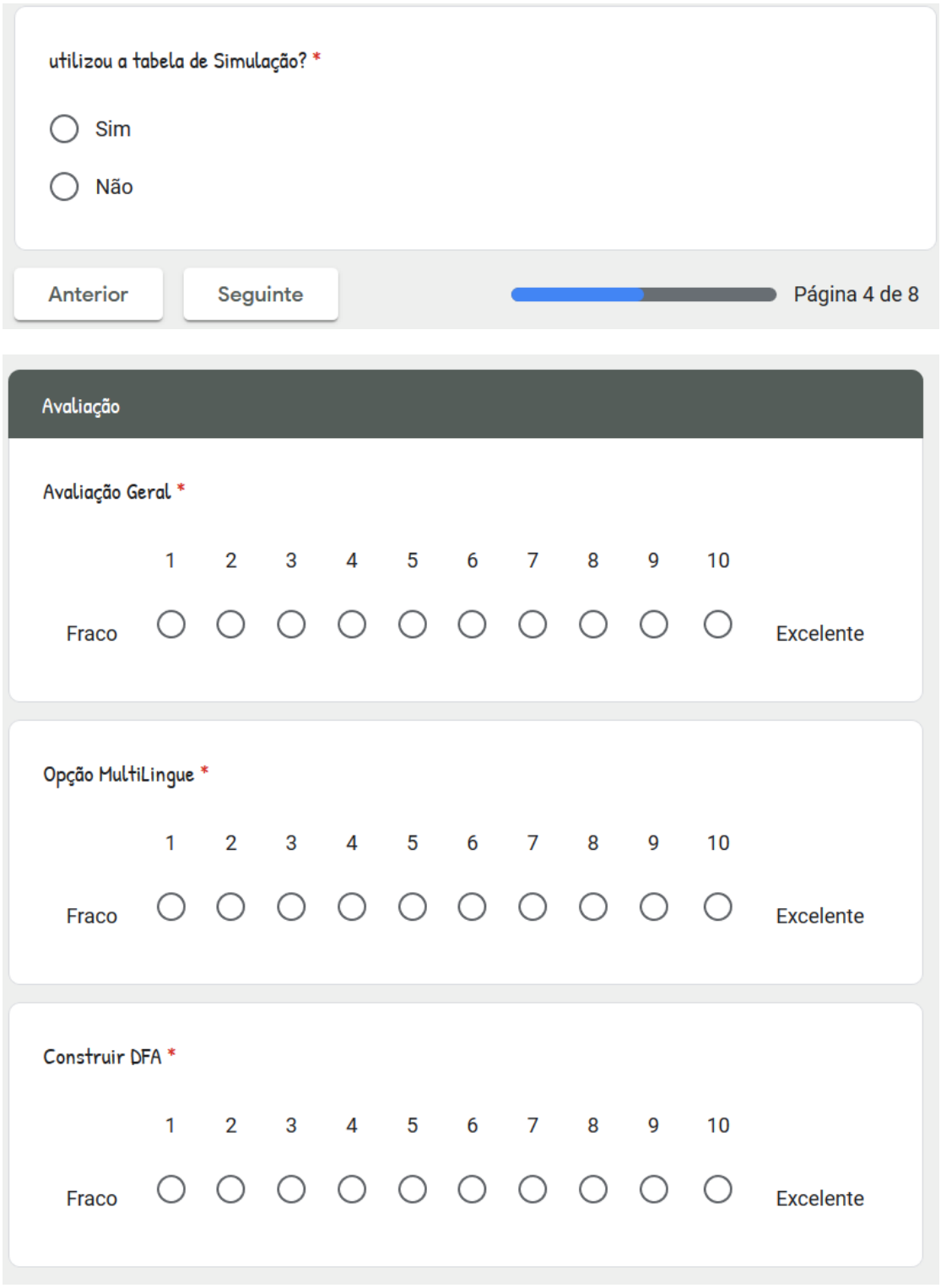
- ◯ Sim
- ◯ Não

utilizou Construir DFA para o grafo? *

- ◯ Sim
- ◯ Não

Simulou passo-a-passo? *

- ◯ Sim
- ◯ Não


utilizou a tabela de Simulação? *
Sim
Não
Anterior
Seguinte
Página 4 de 8
Avaliação
Avaliação Geral *
1 2 3 4 5 6 7 8 9 10
Fraco
Excelente
Opção MultiLingue *
1 2 3 4 5 6 7 8 9 10
Fraco
Excelente
Construir DFA *
1 2 3 4 5 6 7 8 9 10
Fraco
Excelente

Simulação passo-a-passo *

	1	2	3	4	5	6	7	8	9	10	
Fraco	○	○	○	○	○	○	○	○	○	○	Excelente

Tabela Simulação *

	1	2	3	4	5	6	7	8	9	10	
Fraco	○	○	○	○	○	○	○	○	○	○	Excelente

Encontrou falhas ou tem propostas melhoria? *

- ○ Sim
- ○ Não

Anterior Seguinte Página 5 de 8

Falhas ou propostas de melhoria

Nesta secção poderá descrever as falhas e propostas de melhoria, e adicionar ficheiros de suporte

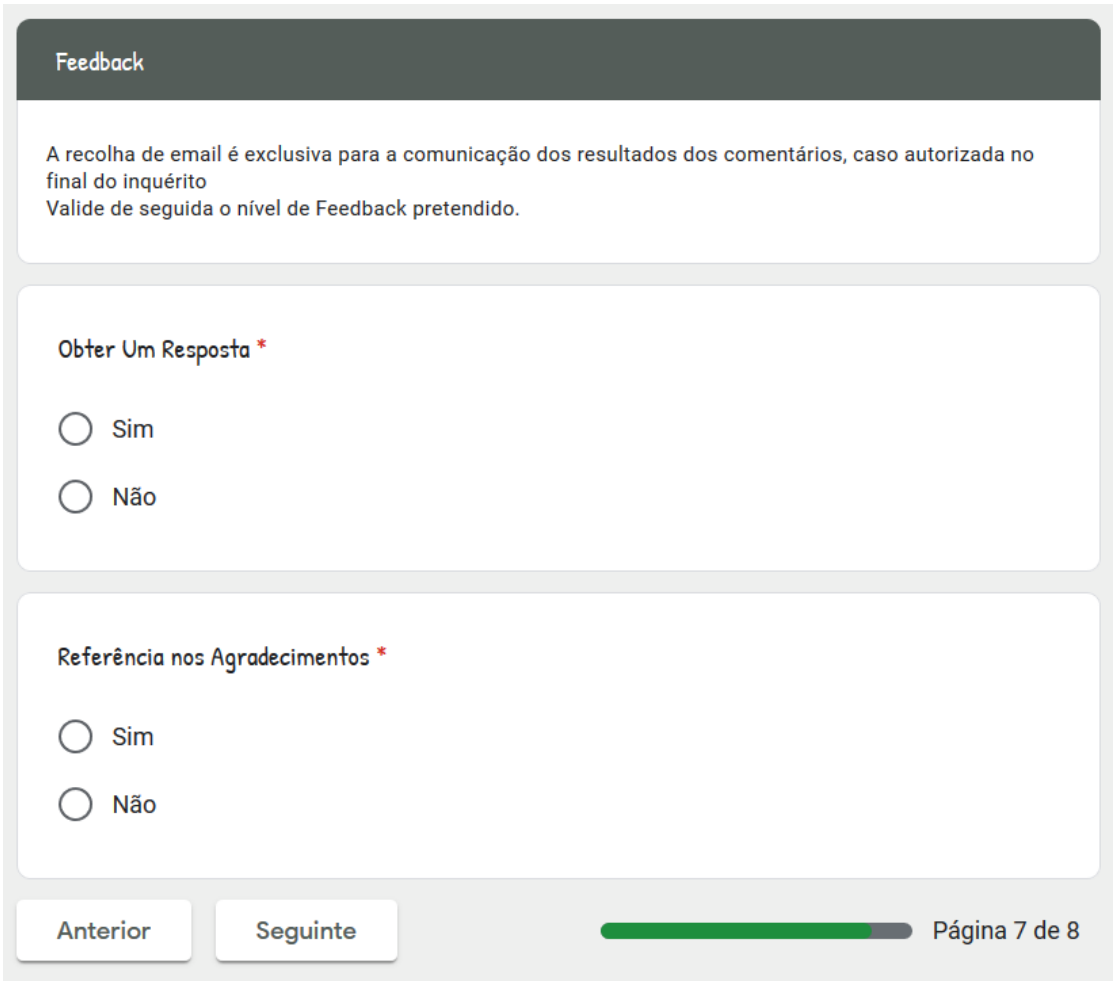
Falha e/ou proposta de melhoria *

A sua resposta

Anexar Documento

Adicionar ficheiro

Anterior Seguinte Página 6 de 8



Feedback

A recolha de email é exclusiva para a comunicação dos resultados dos comentários, caso autorizada no final do inquérito
Valide de seguida o nível de Feedback pretendido.

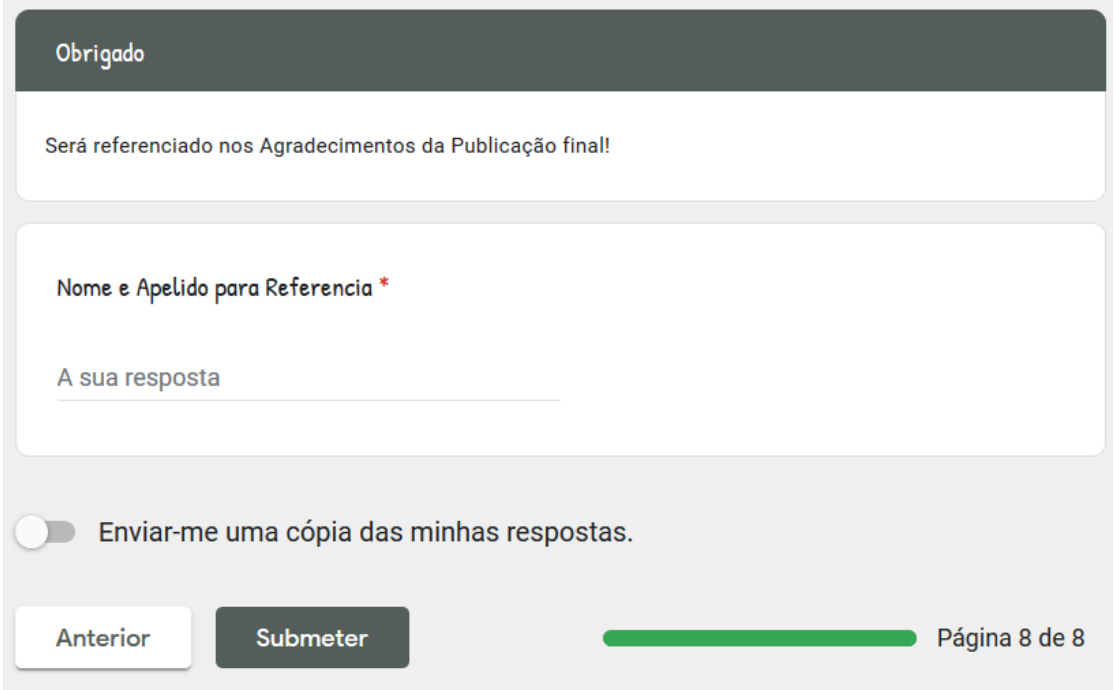
Obter Um Resposta *

- Sim
- Não

Referência nos Agradecimentos *

- Sim
- Não

Anterior Seguinte Página 7 de 8



Obrigado

Será referenciado nos Agradecimentos da Publicação final!

Nome e Apelido para Referencia *

A sua resposta

Enviar-me uma cópia das minhas respostas.

Anterior Submeter Página 8 de 8