Licenciatura em Engenharia Informática – DEI/ISEP
Linguagens e Programação 2021/2022
Aulas Teórico-Práticas

Ficha TP 1

Expressões Regulares e Autómatos Finitos

Objetivos:

- Estudo de conceitos, modelos, técnicas e ferramentas que compõem a Teoria das Linguagens Formais
- Introdução ao reconhecimento de padrões e Expressões Regulares
- Propriedades e Aplicações de Expressões Regulares
- Introdução ao conceito de Autómato Finito e notações utilizadas na sua representação;
- Classificação de AF quanto ao seu comportamento: Determinísticos (AFD) ou Não Determinísticos (AFN)
- Validação de palavras utilizando Autómatos Finitos;
- Conversão de autómatos finitos não determinísticos (AFN) em autómatos finitos determinísticos (AFD);
- Minimização de Autómatos Finitos Determinísticos;
- Consolidação dos conceitos através da realização de alguns exercícios.

Teoria das Linguagens Formais

Entende-se por Teoria das Linguagens Formais o estudo de modelos matemáticos que possibilitam a especificação e o reconhecimento de linguagens, sua classificação, estrutura, propriedades e características. Tem sofrido uma evolução considerável devido à sua particular adequação na descrição de processos computacionais e linguagens de programação.

Tal como as linguagens naturais, as linguagens formais (assim designadas por serem definidas por um conjunto de formalismos matemáticos abstratos) exprime-se através de símbolos, tal como as linguagens humanas ao longo da história usaram símbolos diversos, desde os hieroglíficos até aos caracteres latinos, passando pelos chineses, gregos e pelos árabes.

É possível definir formalmente a Teoria das Linguagens Formais como o conjunto de conceitos, propriedades, técnicas e ferramentas para descrever e caracterizar as linguagens formais, gerar palavras de determinadas linguagens (gramáticas), reconhecer e perceber palavras de determinadas linguagens (autómatos, expressões regulares, etc.).

1.1 Conceitos Básicos

Alfabeto: Qualquer conjunto finito, não vazio, de símbolos (ou letras). Um alfabeto será designado por ∑. Considerem-se os seguintes exemplos:

$\Sigma = \{0, 1\}$, o alfabeto binário.

$\Sigma = \{a, b, c, \ldots, z\}$, o conjunto das letras minúsculas.

Palavra: ou cadeia, é uma sequência finita (eventualmente vazia) de símbolos de alfabeto Σ. A palavra vazia contém zero ocorrências de símbolos e será denotada por ε. Designa-se por $|W|$ o comprimento da palavra W. Por exemplo, $|aa| = 2$ e $|\varepsilon| = 0$.

Potência de um alfabeto: Σ^k denota o conjunto de todas as palavras de comprimento k, sobre um alfabeto Σ. Exemplos, para o alfabeto $\Sigma = \{0, 1\}$:

$\Sigma^0 = \{\varepsilon\}$ (contém apenas a palavra vazia)
$\Sigma^1 = \{0, 1\}$ (não confundir com Σ)
$\Sigma^2 = \{00, 01, 10, 11\}$
$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

O conjunto de todas as palavras sobre o alfabeto Σ será denotado por $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
Se for excluída a palavra vazia obter-se-á Σ^+, formalmente:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$$
$$\Sigma^* = \Sigma^+ \setminus \{\varepsilon\}$$

Fecho de um alfabeto Σ, representado $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$, corresponde ao conjunto de todas as palavras que podem ser formadas com os símbolos de Σ, incluindo a palavra vazia ε. O fecho positivo de Σ, definido por Σ^+ é definido como $\Sigma^* \setminus \{\varepsilon\}$, ou seja, todas as cadeias formadas com os símbolos de Σ, exceto a palavra vazia. Considere-se o seguinte exemplo, para o alfabeto $\Sigma = \{0,1\}$:

$$\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$$
$$\{0,1\}^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$$

Concatenação de palavras: Justapondo duas palavras α e β será obtido $\alpha\beta$, uma palavra formada pelos símbolos de α seguidos dos símbolos de β. A operação de concatenação de palavras possui as seguintes propriedades:

- A palavra vazia ε constitui o **elemento identidade** (neutro) da operação, isto é, para qualquer palavra α:
  $$\alpha\varepsilon = \varepsilon\alpha = \alpha$$
- É **associativa**, ou seja, para quaisquer palavras α, β e γ $(\alpha\beta)\gamma = \alpha(\beta\gamma)$
- Não é **comutativa**, ou seja, $\alpha\beta \neq \beta\alpha$ sempre que , $\alpha \neq \beta$. Considere-se o exemplo:

  $\alpha = aa$
  $\beta = b$
  $\alpha\beta = aab \neq baa = \beta\alpha$

- O comprimento é **aditivo** relativamente à concatenação, $|\alpha\beta| = |\alpha| + |\beta|$
- Duas palavras são iguais sempre que uma for a cópia, letra a letra, da outra;
- Dadas duas palavras α e β, a palavra α é **prefixa** da palavra $\alpha\beta$, e a palavra β é **sufixo** da palavra $\alpha\beta$;
- Diz-se que a palavra α é parte (**subpalavra**) da palavra β sempre que existam as palavras γ e ψ, tais que $\beta = \gamma\alpha\psi$;

- Para qualquer número natural n e qualquer palavra α, designa-se por α^n a **potência** de ordem n. Considere-se o exemplo, para o alfabeto $\sum = \{a, b, c, \ldots, z\}$ e α = ab teremos:

$$\alpha^1 = ab$$
$$\alpha^2 = abab$$
$$\alpha^3 = ababab$$

  Note-se que $(ab)^3$ = ababab, enquanto que ab^3 = abbb. Por convenção para qualquer palavra α temos que $\alpha^0 = \varepsilon$.

- Designa-se por α^R a **palavra inversa** de α, ou seja, se α = ab então α^R = ba. Desta forma: $(\alpha^R)^R = \alpha$ e $(\alpha^R)^n = (\alpha^n)^R$ para qualquer n.

1.2 Linguagens Formais

Uma linguagem formal é um conjunto de cadeias de símbolos (palavras) de determinado alfabeto. Isto é, uma linguagem sobre o alfabeto $\sum$ é um subconjunto de $\sum^*$. Assim, por exemplo, o conjunto de palavras válidas da língua portuguesa poderia ser definido como um subconjunto de $\{a, b, c, \ldots, z\}^+$. Uma linguagem é finita se as suas frases formam um conjunto finito. Caso contrário, a linguagem é infinita. Uma linguagem infinita precisa ser definida através de uma representação finita.

Por exemplo, a linguagem dos números naturais menores que 10 é finita, e pode ser representada, literalmente, como L = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Já a linguagem dos números naturais como um todo não é uma linguagem finita, já que existem infinitos números naturais. Porém, existem mecanismos que permitem expressar esta e outras linguagens infinitas através de representações finitas. É o caso das gramáticas e das expressões regulares.

Uma linguagem **L** sobre um alfabeto $\sum$ é um conjunto de palavras de $\sum^*$, isto é, é um subconjunto de $\sum^*$ ($L \subseteq \sum^*$). Uma linguagem diz-se **finita** se for um conjunto finito. Relativamente a linguagem **L** diz-se que:

- L pode não incluir todas as palavras de $\sum^*$, nem sequer as suas palavras incluírem todos os símbolos de $\sum$;
- Uma **linguagem vazia**, que não contém palavras, designa-se por $\varnothing$. Note-se que não se trata da mesma da linguagem definida por $\{\varepsilon\}$;
- A linguagem designa-se como **completa** se coincide com a totalidade do conjunto $\sum^*$;
- Se L e M são linguagens do alfabeto $\sum$, a concatenação de L com M é uma linguagem do alfabeto $\sum$, definida por $LM \{\alpha\beta | \alpha \in L \text{ e } \beta \in M\}$. Por exemplo: {a, ab}{b, ba} = {ab, aba, abb, abba};
- As potências L^n de uma linguagem L são definidas indutivamente por:

$$L^0 = \{\varepsilon\}$$
$$L^{n+1} = LL^n\text{, para } n \geq 1$$
$$L^n = \{x_1 x_2 \ldots x_n | x_i \in L, 1 \leq i \leq n\}$$

  Isto é, a linguagem das palavras obtidas por concatenação de n palavras de L. Considere-se o seguinte exemplo em que L = {ab, aab}:

$$\{ab, aab\}^0 = \{\varepsilon\}$$
$$\{ab, aab\}^1 = \{ab, aab\}$$

$\{ab, aab\}^2 = \{abab, abaab, aabab, aabaab\}$
$\{ab, aab\}^3 = \{ ababab, ababaab, abaabab, aababab, abaabaab, aababaab, \ldots\}$

- O **fecho de Kleene** de uma linguagem L é a reunião de todas as potências finitas de L. Note-se que $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots = \cup_{n \geq 0} L^n$, ou de forma equivalente, corresponde ao conjunto das palavras que se podem formar por concatenação de zero ou mais palavras de L. Refira-se ainda que Σ^*, definido como o conjunto de todas as palavras do alfabeto Σ, não é mais do que o fecho de Kleene de Σ. Considerem-se os seguintes exemplos:

  $\{1\}^* = \{\varepsilon, 1, 11, 111, 1111, 11111, \ldots\} = \{1^n | n \in N\}$
  $\{01\}^* = \{\varepsilon, 01, 0101, 010101, 01010101, 0101010101, \ldots\}$
  $\{000\}^* = \{\varepsilon, 000, 000000, 000000000, \ldots\} = \{0^{3n} | n \in N\}$
  $\{000, 00000\}^* = \{\varepsilon, 000, 00000, 000000\} \cup \{0^n | n \geq 8\}$
  $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots\}$

- O **fecho positivo** da linguagem L, definido por $L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots$, corresponde ao conjunto das palavras construídas pela concatenação de qualquer número de palavras de L, excluindo a palavra vazia ε.

  – Se $\varepsilon \in L$ então $L^+ = L^*$

  – Se $\varepsilon \notin L$ então $L^+ = L^* \backslash \{\varepsilon\}$

- Toda a expressão regular representa uma linguagem regular;

- Toda a linguagem regular é representável por alguma expressão regular. Considere o exemplo, o alfabeto binário $\Sigma = \{0, 1\}$. Os seguintes conjuntos são linguagens:

  – é uma linguagem vazia;

  – $\{\varepsilon\}$ é uma linguagem constituída pela palavra nula;

  – $\{0\}$ é uma linguagem constituída por uma única palavra;

  – $\{00, 01, 10, 11\}$ é uma linguagem constituída por palavras de comprimento 2.

1.3 Expressões Regulares

As expressões regulares (ER) são regras bem definidas que permitem gerar um conjunto de sequências de símbolos (palavras). Uma expressão regular traduz um conjunto de padrões, possivelmente complicados e difíceis pela sua dimensão de enumerar, numa expressão de dimensão curta e relativamente fácil de interpretar.

Uma expressão regular constrói-se com base num conjunto de meta-caracteres que são interpretados de forma especial, escritos numa dada sequência e possivelmente intercalados com sequências de caracteres sem significado especial que não seja a sua verificação. O conjunto de palavras geradas por uma expressão regular designa-se uma linguagem regular.

As palavras geradas por uma expressão regular são construídas com caracteres de um conjunto de símbolos designado alfabeto Σ. A linguagem regular gerada por uma expressão regular *r* tem a designação $L(r)$, constituída por sequências de símbolos pertencentes ao alfabeto. A linguagem regular vazia (sem qualquer *string*) designa-se por $\emptyset$. É de notar que, a linguagem vazia $\emptyset$ é diferente da linguagem que só possui a *string* vazia (que se designa por ε) $L(\varepsilon) = \{\varepsilon\}$. O conjunto de expressões regulares sobre um alfabeto $\sum$ e o conjunto das linguagens por elas descritas são definidos indutivamente por:

- é uma expressão regular sobre o alfabeto $\sum$ e descreve a linguagem $\{\varepsilon\}$, $L(\varepsilon) = \{\varepsilon\}$;
- é uma expressão regular sobre o alfabeto $\sum$ e descreve a linguagem vazia $\emptyset$ isto é, $L(\emptyset) = \emptyset$;
- Se $a \in \sum$ então a é uma expressão regular sobre $\sum$ e descreve a linguagem $\{a\}$, isto é, $L(a) = \{a\}$;
- Se r e s são expressões regulares sobre $\sum$ que descrevem as linguagens $L(r)$ e $L(s)$:

  $(r|s) = L(r) \cup L(s)$
  $(rs) = L(r)L(s)$
  $(r^*) = L(r)^*$

  Considere-se o seguinte exemplo, r e s que permitem reconhecer as palavras, {aa, ab} e {bb, ba}, respetivamente. Note-se que a linguagem reconhecida pela expressão r representa-se por L(r) = {aa, ab}, de igual modo, a linguagem reconhecida pela expressão s representa-se por L(s) = {bb, ba}. Assim:

  União
  $r \mid s = L(r) \cup L(s) = \{aa, ab, bb, ba\}$

  Concatenação
  $rs = L(r)L(s) = \{aabb, aaba, abbb, abba\}$

  Fecho de Kleene
  $r^* = L(r)^* = \{\varepsilon, aa, ab, aaaa, aaab, abaa, abab, aaaaaa, aaaaab, \ldots\}$

- Qualquer expressão regular r representa uma linguagem que se designa por $L(r)$;
- Diz-se que duas expressões regulares são **equivalentes** se e só se as linguagens por elas descritas são iguais.

Além das operações de união, intersecção e potência, herdadas da teoria dos conjuntos, existem outras operações sobre linguagens relacionadas com a operação de concatenação de palavras. Referem-se o produto de duas linguagens e o fecho de uma linguagem, que, juntamente com a união, constituem as chamadas operações regulares. A importância das operações regulares advém do facto de elas serem utilizadas na definição das linguagens regulares.

As regras de precedência para as operações união (|), concatenação e fecho de Kleene (*) correspondem às usadas nas expressões aritméticas para a adição, a multiplicação e a potenciação, respetivamente. Isto é, a precedência é dada por **fecho de Kleene > concatenação > união**.

Considerem as expressões regulares r, s e t, desta forma a expressão regular $rs^*|t$ será avaliada da forma seguinte:

1º Passo - avaliação da potência (fecho de Kleene) rs*|t ⇒ r(ε|s|ss|sss|…)|t

2º Passo - avaliação da concatenação

rs*|t ⇒ (rε|rs|rss|rsss|…)|t   sendo que rε = r

3º Passo - avaliação da união

rs*|t ⇒ r|rs|rss|rsss|…|t

Exemplo: Seja ∑ = {0, 1}

Expressão Regular	Linguagem Descrita
(0|1)	{0, 1}
(0*)	{ε, 0, 00, 000, 0000, 00000, …}
((0*)(11))	{11, 011, 0011, 00011, 000011, 0000011, …}
((01)*)	{ε, 01, 0101, 010101, 01010101, 0101010101, …}
((0|1)*)	{ε, 0, 1, 00, 10, 000, 100, 010, 001, 110, 101, 011, …}
((1*)(0((0|1)*)))	{0, 10, 01, 00, 000, 100, 010, 001, 110, 101, 011, …}
(0|1)*0	sequências de palavras terminadas (ou com sufixo) em 0
(0|1)*(1)	sequências de palavras terminadas (ou com sufixo) em 1
(0|1)*101(0|1)*	sequência que contem o factor ou subpalavra 101

Propriedades das Expressões Regulares

As propriedades das expressões regulares servem para simplificar expressões regulares e provar a equivalência entre duas expressões regulares. Sejam u, v e x expressões regulares. Tem-se que:

```
     u|v   =  v|u
     u|u   =  U
 (u|v)|x   =  u|(v|x)
      uε   =  εu = u
   (uv)x   =  u(vx)
  u(v|x)   =  uv|ux
  (u|v)x   =  ux|vx
      ε*   =  ε
      u*   =  u*u* = (u*)* = u*|u*
      u*   =  ε|u*| = (ε|u)u* = ε|uu*
      u*   =  (u|…|u^k)*, k≥1
      u*   =  ε|u| …|u^(k-1)|u^k u*, k>1
     u*u   =  uu*
  (u|v)*   =  (u*|v*)* = (u*v*)* = (u*v)*u* = u*(vu*)*
  u(vu)*   =  (uv)*u
  (u*v)*   =  ε|(u|v)*v
  (uv*)*   =  ε|u(u|v)*
```

Aplicações de Expressões Regulares

Em diferentes aplicações e linguagens de programação, usam-se variantes das expressões regulares, denominados por padrões, para descrever e reconhecer conjuntos de palavras que correspondem a linguagens regulares, como por exemplo:

- Em linguagens de programação que manipulam strings: bash, tcl, perl, python, etc.;

- Reconhecedores de padrões em ficheiros, como o `grep`: `grep 'ex*' teste.txt`;
- Analisadores lexicais, como o **FLEX**, que permitem seccionar um texto em elementos identificados (*tokens*): por exemplo num programa em C, identificar o que são palavras reservadas (**`if`**, **`while`**, **`struct`**, . . .), variáveis, constantes, operadores, etc. São usados pelos compiladores;
- Editores de texto, como o **`emacs`**, na procura de palavras.

Um padrão é uma sequência de símbolos que representa uma linguagem num dado alfabeto. O conjunto de palavras que são reconhecidas por um dado padrão denota-se por $L(\alpha)$. Os padrões normalmente usados são os da tabela seguinte:

Tabela 1.1 - Padrões normalmente usados

Padrão	Descrição
x	O carácter "x"
.	Qualquer carácter excepto mudança de linha
\n	Mudança de linha
[xyz]	Um dos caracteres "x", "y", "z"
xyz	A cadeia de caracteres xyz
[a-zA-Z]	Um dos caracteres no intervalo de "a" a "z" ou de "A" a "Z"
[-+*/]	Qualquer um dos operadores "-", "+", "*" ou "/", sendo que o símbolo "-" tem de aparecer em primeiro lugar dada a possibilidade de ambiguidade com a definição de intervalo
[abj-oZ]	Um dos caracteres "a", "b" ou de "j" a "o" ou "Z"
[^A-Z\n]	Qualquer caracter excepto no intervalo de "A" a "Z" ou mudança de linha
r*	O caracter "r" zero ou mais vezes
r+	O caracter "r" uma ou mais vezes
r?	O caracter "r" zero ou uma vez
r{2, 5}	O caracter "r" repetido de duas a cinco vezes
r{2, }	O caracter "r" repetido pelo menos duas vezes
r{4}	O caracter "r" repetido exactamente quatro vezes
{macro}	Substituição/Expansão da macro definida anteriormente
(r)	O caracter "r", sendo que os parêntesis permitem estipular precedências
xyz*	A sequência "xy" seguida de zero ou mais "z"
(xyz)*	A sequência "xyz" repetida zero ou mais vezes
r|s	O caracter "r" ou "s" (alternativa)
^r	O caracter "r" apenas se no início da linha
r$	O caracter "r" apenas se no final da linha (não consome o \n)
^xyz$	Uma linha que contém apenas a cadeia de caracteres "xyz"
<<EOF>>	Fim de ficheiro

Exemplo:
Definição de um identificador:

letra $\rightarrow$ [A-Za-z]
dígito $\rightarrow$ [0-9]
identificador $\rightarrow$ {letra} ({letra}|{digito})*

Definição de um número com parte decimal

dígito $\rightarrow$ [0-9]
dígitos $\rightarrow$ {dígito}+
parteFraccionária $\rightarrow$ (\. {dígitos})?
expoente $\rightarrow$ (E [+-]? {dígitos})?
num $\rightarrow$ {dígitos} {parteFraccionária} {expoente}

Bibliografia:

[1]. Jeffrey D. Ullman, E. Hopcroft, Rajeev Motwani, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 2nd Edition, 2001.

[2]. Rui Gustavo Crespo, Processadores de Linguagens - da concepção à implementação, IST Press, 2001.

Autómatos finitos

Considere um interruptor comum que tem dois estados possíveis: **on** e **off** (o estado **off** é o estado inicial). Sobre este interruptor podem ser realizadas duas ações: **ligar** e **desligar**. O funcionamento deste interruptor pode ser descrito através do autómato finito apresentado na figura 2.1.

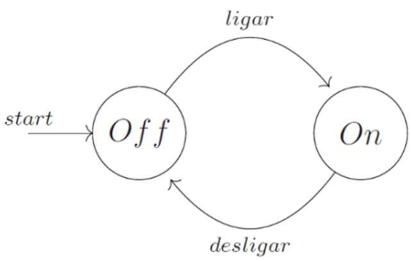

Figura 2.1: Autómato de um interruptor

Um autómato finito ou máquina de estados, é um formalismo, que permite representar de forma clara, um qualquer processo composto por um conjunto de estados, e transições entre esses estados. Mais formalmente um autómato finito (AF) é representado por um tuplo A = (S, ∑, s_0, F, δ), no qual:

- S é um conjunto finito de estados não vazio;
- ∑é o alfabeto de entrada, um conjunto finito de símbolos não vazio;
- s_0 é o estado inicial, um elemento de S;
- F é conjunto de estados finais. F é um subconjunto de S;
- δ é a função de transição, recebe como argumentos um estado e um símbolo de entrada, e devolve um novo estado (eventualmente o mesmo): $\delta : S \times \sum \rightarrow S$.

Considere-se um **AF** capaz de processar números binários terminados em "10" (equivalente à expressão regular "(0|1)*10").

$A = (\{s_0, s_1, s_2\}, \{0, 1\}, s_0, \{s_2\}, \delta)$

onde,

$\delta\ (s_0, 0) = \{s_0\}$

$\delta\ (s_0, 1) = \{s_0, s_1\}$

$\delta\ (s_1, 0) = \{s_2\}$

Este autómato finito é representável por uma tabela de transições como a seguir apresentado (ver tabela 2.1).

Tabela 2.1: Tabela de transições do AF

	0	1
$\rightarrow s_0$	$\{s_0\}$	$\{s_0, s_1\}$
s_1	$\{s_2\}$	$\varnothing$
$*s_2$	$\varnothing$	$\varnothing$

Este autómato finito também pode ser representado graficamente, como se exemplifica na figura 2.2, seguindo a simbologia explicitada na tabela 2.2.

Tabela 2.2: Simbologia da representação gráfica de um AF

	Estado inicial s_0
	Estado s_1
	Transição que consome o símbolo 0
	Estado final s_2

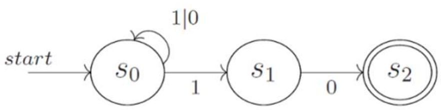

Figura 2.2: Representação gráfica do AF

Aquando do processamento de uma frase por um **AF** acontece uma das seguintes situações:

- Após processar o último símbolo está num estado final: o autómato pára e a frase é aceite;
- Após processar o último símbolo está num estado não final: o autómato pára e a frase rejeitada;
- O autómato está num estado em que para o símbolo seguinte não existe função de transição: o autómato pára e a frase rejeitada.

1.4 Classificação de Autómatos Finitos

Um autómato finito diz-se **determinístico** (**AFD**) se, em cada um dos seus estados e perante um símbolo, puder transitar para um único estado, i.e.,

$$\forall\ s_1, s_2, s_3, a: (s_1, a, s_2) \in \delta \wedge (s_1, a, s_3) \in \delta \rightarrow s_2 = s_3$$

Caso contrário, diz-se **não determinístico** (**AFN**).

No que diz respeito à comparação entre **AFD**s e **AFN**s pode-se acrescentar que:

- Os **AFD** são mais rápidos (tempo de computação) que os **AFN**;
- Os **AFN** ocupam muito menos espaço (têm menos estados) que os **AFD**;
- Um **AFD** é um caso particular de **AFN** em que para qualquer estado **K** e qualquer símbolo de entrada x, só pode haver uma transição a partir de **K**.

1.5 Exercícios Resolvidos

1. Represente graficamente e através da tabela de transições, os AFN capazes de reconhecer a seguinte linguagem. Uma string no alfabeto {a, b} em que todas as strings acabam em “b” e começam em “a”.

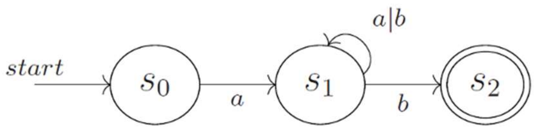

Figura 2.3: Representação gráfica do AF

Tabela 2.3: Tabela de transições do AF

	a	B
$\rightarrow s_0$	$\{s_1\}$	$\varnothing$
s_1	$\{s_1\}$	$\{s_1, s_2\}$
$*s_2$	$\varnothing$	$\varnothing$

A representação formal do autómato é:

$$A = (\{s_0, s_1, s_2\}, \{a, b\}, s_0, \{s_2\}, \delta$$

onde:

$\delta(s_0, a) = \{s_1\}$

$\delta(s_1, a) = \{s_1\}$

$\delta(s_1, b) = \{s_1, s_2\}$

2. Considerando o alfabeto $\sum = \{A, B, C, \ldots Z, a, b, c, \ldots z\}$, represente a seguinte linguagem utilizando um AFN, $L(A) = \{u \mid \in \sum^* :$ u tem sempre pelo menos dois *a*'s juntos}.

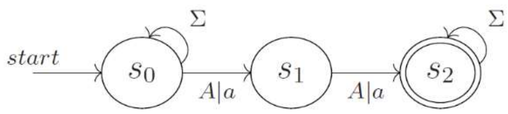

Figura 2.4: Representação gráfica do AF

O exercício podia ser resolvido mediante a representação formal do autómato:

$A = (\{s_0, s_1, s_2\}, \{A, B, C, \ldots, Z, a, b, c, \ldots, z\}, s_0, \{s_2\}, \delta)$

onde δ, é definido pela seguinte tabela de transições:

	$\sum \backslash \{A,a\}$	$\{A, a\}$
$\rightarrow s_0$	$\{s_0\}$	$\{s_0, s_1\}$
s_1	$\varnothing$	$\{s_2\}$
$*s_2$	$\{s_2\}$	$\{s_2\}$

1.6 Validação de palavras utilizando Autómatos Finitos

Considere o autómato finito não determinístico que permite reconhecer números binários terminados em “10”.

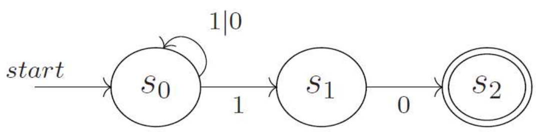

Figura 2.5: Representação gráfica do AF

Conforme descrito anteriormente, formalmente este autómato é definido da seguinte forma:

$A = (S, \sum, s_0, F, \delta) = (\{s_0, s_1, s_2\}, \{0, 1\}, s_0, \{s_2\}, \delta)$

onde,

$\delta(s_0, 0) = \{s_0\}$

$\delta(s_0, 1) = \{s_0, s_1\}$

$\delta(s_1, 0) = \{s_2\}$

Para verificar formalmente se uma determinada palavra é reconhecida pelo autómato, é necessário explorar todos os caminhos a partir do estado inicial. Para este fim é utilizada a transição estendida $\hat{\delta}$ que calcula o conjunto de estados passíveis de serem atingidos a partir de um determinado estado, após processar uma sequência de símbolos do alfabeto.

Note-se que a transição $\hat{\delta}$ $(s_0; \varepsilon) = \{s_0\}$ indica que no caso de não ser consumido qualquer símbolo (representado por ε), não haverá lugar à transição de estado. O reconhecimento da palavra “100110” é descrito através da seguinte sequência:

$\hat{\delta}\ (s_0,\varepsilon) = \{s_0\}$

$\hat{\delta}\ (s_0,1) = \delta\{s_0, 1\} = \{s_0, s_1\}$

$\hat{\delta}\ (s_0,10) = \delta\{s_0, 0\} \cup \delta(s_1, 0) = \{s_0\} \cup \{s_2\} = \{s_0, s_2\}$

$\hat{\delta}\ (s_0,100) = \delta\{s_0, 0\} \cup \delta(s_2, 0) = \{s_0\} \cup \varnothing = \{s_0\}$

$\hat{\delta}\ (s_0,1001) = \delta\{s_0, 1\} = \{s_0, s_1\}$

$\hat{\delta}(s_0, 10011) = \delta\{s_0, 1\} \cup \delta(s_1, 1) = \{s_0, s_1\} \cup \emptyset = \{s_0, s_1\}$

$\hat{\delta}(s_0, 100110) = \delta\{s_0, 0\} \cup \delta(s_1, 0) = \{s_0\} \cup \{s_2\} = \{s_0, s_2\}$

A representação gráfica da sequência de transições é apresentada na figura 2.6. Relembre-se que, o processamento bem-sucedido da palavra implica o consumo integral de todos os símbolos que a compõe, e que após o consumo do último símbolo o autómato se encontre num estado final. Assim, a palavra em questão ("100110") poderia ser integralmente consumida no estado s_0, no entanto, como s0 não se trata de um estado final esse caminho não seria válido, sendo apenas válido o caminho terminado em s_2.

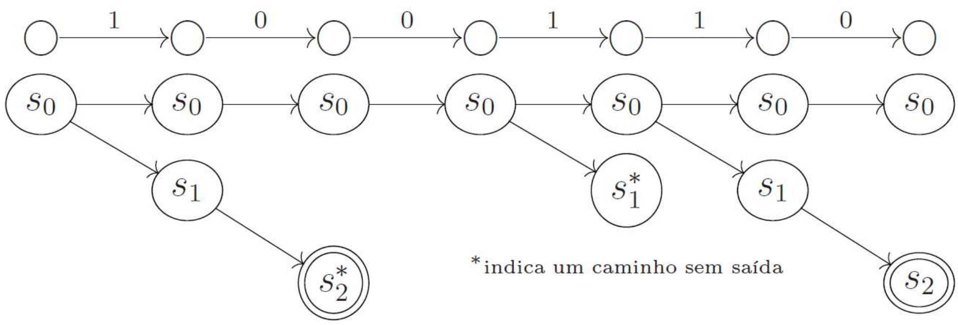

Figura 2.6: Representação gráfica da análise duma palavra pelo AF

1.7 Conversão de um AFN num AFD

A diferença mais relevante entre um AFN e um AFD consiste no facto de que, num AFD é sempre possível determinar qual é o estado para que o autómato transita após o consumo de um qualquer símbolo, pois o retorno da função de transição num AFD é um estado único, enquanto num AFN é um conjunto de estados.

Este facto é verificável graficamente no autómato (não determinístico) da figura 2.5, pois no estado s0 o consumo do símbolo "1" não determina objetivamente para que estado o autómato transitaria. Alternativamente este facto também é verificável na tabela de transições, pois o resultado dos pares (estado, símbolo) é um conjunto de estados, conforme se constata na tabela 2.4.

Tabela 2.4: Tabela de transições do AFN

	0	1
$\rightarrow s_0$	$\{s_0\}$	$\{s_0, s_1\}$
s_1	$\{s_2\}$	$\emptyset$
$*s_2$	$\emptyset$	$\emptyset$

Conversão de um AFN num AFD

De forma a converter um AFN num AFD, é necessário converter todos os conjuntos de estados em estados únicos. Desta forma, os *n* estados do AFN (tabela 2.4) darão lugar a 2^n estados no AFD (conforme tabela 2.5).

Tabela 2.5: Tabela de transições do AFD

	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\rightarrow\{s_0\}$	$\{s_0\}$	$\{s_0, s_1\}$
$\{s_1\}$	$\{s_2\}$	$\varnothing$
$*\{s_2\}$	$\varnothing$	$\varnothing$
$\{s_0, s_1\}$	$\{s_0, s_2\}$	$\{s_0, s_1\}$
$*\{s_0, s_2\}$	$\{s_0\}$	$\{s_0, s_1\}$
$*\{s_1, s_2\}$	$\{s_2\}$	$\varnothing$
$*\{s_0, s_1, s_2\}$	$\{s_0, s_2\}$	$\{s_0, s_1\}$

Nesta tabela a primeira coluna contém todas as combinações de estados possíveis para um autómato com n estados. Como os conjuntos que contenham um estado final, neste caso s_2, serão finais, devem por isso ser anotados em conformidade. As restantes colunas, cada uma das células contem a reunião dos estados para os quais o autómato transitaria estando em qualquer um dos estados apresentados na primeira coluna com o símbolo apresentado na primeira linha da matriz. Por exemplo, a transição $\hat{\delta}$ $(\{s_0, s_1\},0)$ é calculada através de $\delta(s_0,0) \cup \delta(s_1, 0) = \{s_0\} \cup \{s_2\}$ $\{s_0, s_2\}$

No passo seguinte, cada conjunto de estados será representado univocamente por um identificador, conforme se apresenta na tabela 2.6.

Tabela 2.6: Estados do AFD renomeados

	0	1
A	A	A
$\rightarrow$ B	B	E
C	D	A
* D	A	A
E	F	E
* F	B	E
* G	D	A
* H	F	E

Nesta fase, a partir do estado inicial “B”, são identificados os estados para os quais é possível transitar, neste caso B e E. Após o que, de forma iterativa, se procede à identificação dos estados para os quais se pode transitar a partir destes últimos, obtendo-se primeiro F e E, e posteriormente B e E a partir de F. Serão estas as únicas linhas a considerar da tabela, pois todas as outras correspondem a transições impossíveis. O resultado da aplicação deste algoritmo é apresentado na tabela 2.7.

Tabela 2.7: Tabela de transições final do AFD

	0	1
$\rightarrow$ B	B	E

E	F	E
* F	B	E

Alternativamente, a tabela 2.5 poderia ter sido construída de forma iterativa a partir do estado inicial, expandindo apenas as transições válidas, evitando-se assim a representação de todos os estados inatingíveis. Isto é:

- Do estado $\{s_0\}$ são atingíveis os estados $\{s_0, s_1\}$ e o próprio ($\{s_0\}$);
- Expandindo o estado $\{s_0, s_1\}$ podemos chegar aos estados $\{s_0, s_2\}$ e ao próprio ($\{s_0, s_1\}$);
- Expandindo o estado $\{s_0, s_2\}$ podemos chegar aos estados $\{s_0\}$ e $\{s_0, s_1\}$.

Tabela 2.8: Tabela simplificada da conversão de um AFN num AFD

	0	1
$\rightarrow \{s_0\}$	$\{s_0\}$	$\{s_0, s_1\}$
$\{s_0, s_1\}$	$\{s_0, s_2\}$	$\{s_0, s_1\}$
$*\{s_0, s_2\}$	$\{s_0\}$	$\{s_0, s_1\}$

O resultado destas iterações é apresentado na tabela 2.8. A representação gráfica do AFD obtido é apresentada na figura 2.7.

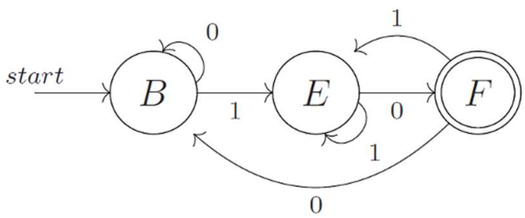

Figura 2.7: Representação gráfica do AFD

Conversão de um AFN num AFD(simplificada)

A versão simplificada do algoritmo de conversão formal de um AFN em AFD, consiste em partindo do estado inicial, a tabela de transições vai sendo construída de forma incremental:

Tabela 2.9: Tabela de transições do AFN

	0	1
$\rightarrow s_0$	$\{s_0\}$	$\{s_0, s_1\}$
s_1	$\{s_2\}$	$\varnothing$
$*s_2$	$\varnothing$	$\varnothing$

1. Copiar estado inicial para a 1ª linha da tabela
2. Sempre que aparecer um novo conjunto, criá-lo na tabela com um novo estado

Tabela 2.10: Tabela de transições do AFN

	0	1
$\rightarrow \{s_0\}$	$\{s_0\}$	$\{s_0, s_1\}$
$\{s_0, s_1\}$	$\{s_0, s_2\}$	$\{s_0, s_1\}$

*$\{s_0, s_2\}$	$\{s_0\}$	$\{s_0, s_1\}$

3. Criar um nome para cada estado

Tabela 2.11: Tabela de transições do AFN

	0	1
$\rightarrow \{s_0\}$ A	$\{s_0\}$	$\{s_0, s_1\}$
$\{s_0, s_1\}$ B	$\{s_0, s_2\}$	$\{s_0, s_1\}$
*$\{s_0, s_2\}$ C	$\{s_0\}$	$\{s_0, s_1\}$

3. Substituir nomes nas transições

Tabela 2.12: Tabela de transições do AFN

	0	1
$\rightarrow \{s_0\}$ A	$\{s_0\}$ A	$\{s_0, s_1\}$ B
$\{s_0, s_1\}$ B	$\{s_0, s_2\}$ C	$\{s_0, s_1\}$ B
*$\{s_0, s_2\}$ C	$\{s_0\}$ A	$\{s_0, s_1\}$ B

4. Eliminar nomes antigos

Tabela 2.13: Tabela de transições do AFN

	0	1
$\rightarrow$ A	A	B
B	C	B
*C	A	B

O resultado da aplicação da versão simplificada do algoritmo de conversão é apresentado na tabela 2.13. A representação gráfica do AFD obtido é similar ao da figura 2.7.

1.8 Minimização de Autómatos Finitos Determinísticos

O processo de minimização para um determinado AFD, pretende calcular de forma expedita o menor autómato possível **equivalente**. Considere-se, por exemplo, o autómato representado na figura 2.8, na exemplificação do algoritmo de simplificação.

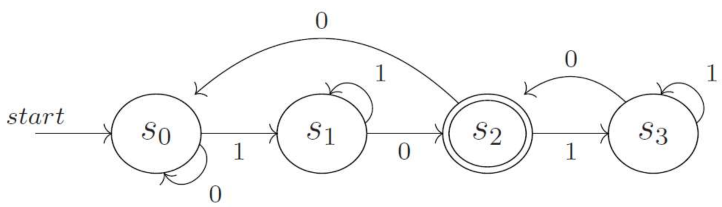

Figura 2.8: Representação gráfica do AFD a simplificar

Inicialmente é necessário construir a tabela de transições do autómato dividindo-a em dois grupos distintos, os estados finais e os não finais (ver tabela 2.9b).

Tabela 2.9: Tabela de transições do AFD, 1ª iteração

(a) Tabela original

	0	1
$\rightarrow s_0$	s_0	s_1
s_1	s_2	s_1
$*s_2$	s_0	s_3
s_3	s_2	s_3

(b) Tabela dividida

		0	1
A	$\rightarrow s_0$	s_0	s_1
	s_1	s_2	s_1
	s_3	s_2	s_3
B	$*s_2$	s_0	s_3

(c) Tabela classificada

		0	1
A	$\rightarrow s_0$	s_0 (A)	s_1 (A)
	s_1	s_2 (B)	s_1 (A)
	s_3	s_2 (B)	s_3 (A)
B	$*s_2$	s_0 (A)	s_3 (A)

No passo seguinte, cada grupo é por sua vez dividido em grupos cuja característica é cada símbolo implicar transições para o mesmo grupo. No exemplo da tabela 2.9c, o grupo dos estados não finais (A) pode ser dividido dois grupos, um com transições para A/A (s_0) e outro com transições para B/A (s_1; s_3). O resultado deste passo é apresentado na tabela 2.10a.

Tabela 2.10: Tabela de transições do AFD, 2ª iteração

(a) Tabela após 1ª iteração

	0	1
$\rightarrow s_0$	s_0	s_1
s_1	s_2	s_1
s_3	s_2	s_3
s_2	s_0	s_3

(b) Tabela com grupos

		0	1
A	$\rightarrow s_0$	s_0	s_1
B	s_1	s_2	s_1
	s_3	s_2	s_3
C	$*s_2$	s_0	s_3

(c) Tabela classificada

		0	1
A	$\rightarrow s_0$	s_0 (A)	s_1 (B)
B	s_1	s_2 (C)	s_1 (B)
	s_3	s_2 (C)	s_3 (B)
C	$*s_2$	s_0 (A)	s_3 (B)

Este processo terá de ser repetido enquanto for possível proceder a subdivisões dos grupos existentes. Na tabela 2.10c é possível verificar que o grupo B não pode ser dividido ficando a tabela igual à da iteração anterior.

Tabela 2.11: Tabela de transições do AFD mínimo

	0	1
$\rightarrow s_0$	s_0	s_1
s_1	s_2	s_1
$*s_2$	s_0	s_1

Finalmente, para os grupos que têm mais que um estado, é escolhido de forma arbitrária apenas um. Neste caso, no grupo B (s_1, s_3), será eliminado um dos estados e as referências a esse estado serão substituídas pelo outro estado, obtendo-se assim o autómato apresentado na tabela 2.11 e na figura 2.9.

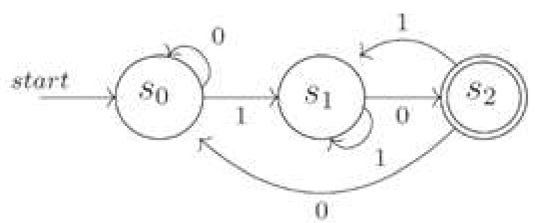

Figura 2.9: Representação gráfica do AFD minimizado

Bibliografia:

[1]. Jeffrey D. Ullman, E. Hopcroft, Rajeev Motwani, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 2nd Edition, 2001.

[2]. Rui Gustavo Crespo, Processadores de Linguagens - da concepção à implementação, IST Press, 2001.