

# Compiladores:

## Apontamentos sobre linguagens regulares

(ano letivo de 2023-2024)

Artur Pereira, Miguel Oliveira e Silva

Abril de 2024

## **Nota prévia**

Este documento representa apenas um compilar de notas sobre a matéria teórico-prática lecionada na unidade curricular “Compiladores”, sem pretensões, por isso, de ser um texto exaustivo. A sua leitura deve apenas ser encarada como ponto de partida e nunca como único elemento de estudo.

# Conteúdo

<b>1</b>	<b>Linguagens</b>	<b>1</b>
1.1	Exemplos de linguagens . . . . .	1
1.2	Elementos básicos sobre linguagens . . . . .	2
1.3	Operações sobre palavras . . . . .	3
1.4	Operações sobre linguagens . . . . .	4
1.4.1	Reunião . . . . .	4
1.4.2	Intersecção . . . . .	4
1.4.3	Diferença e complementação . . . . .	5
1.4.4	Concatenação, potência e fecho de Kleene . . . . .	5
1.4.5	Propriedades das várias operações . . . . .	6
<b>2</b>	<b>Linguagens Regulares (LR) e Expressões Regulares (ER)</b>	<b>7</b>
2.1	Definição de linguagem regular . . . . .	7
2.2	Expressões regulares . . . . .	8
2.3	Propriedades das expressões regulares . . . . .	9
2.4	Simplificação notacional . . . . .	10
2.5	Extensão notacional . . . . .	11
2.6	Aplicação das expressão regulares . . . . .	13
2.7	Exercícios . . . . .	13

<b>3</b>	<b>Gramáticas regulares (GR)</b>	<b>14</b>
3.1	Definição de gramática regular . . . . .	15
3.2	Operações sobre gramáticas regulares . . . . .	15
3.2.1	Reunião de gramáticas regulares . . . . .	15
3.2.2	Concatenação de gramáticas regulares . . . . .	16
3.2.3	Fecho de Kleene de gramáticas regulares . . . . .	17
3.3	Definição de gramática regular generalizada . . . . .	18
<b>4</b>	<b>Equivalência entre ER e GR</b>	<b>19</b>
4.1	Conversão de ER em GR . . . . .	19
4.1.1	Gramáticas regulares dos elementos primitivos . . . . .	20
4.1.2	Algoritmo de conversão . . . . .	20
4.2	Conversão de GR em ER . . . . .	22
4.2.1	Algoritmo de conversão . . . . .	22
<b>5</b>	<b>Autômatos Finitos Deterministas (AFD)</b>	<b>24</b>
5.1	Definição de autômato finito determinista . . . . .	25
5.2	Linguagem reconhecida por um AFD . . . . .	27
5.3	Projeto de um AFD . . . . .	28
5.4	AFD reduzido . . . . .	30
5.4.1	Algoritmo de redução de AFD . . . . .	31
<b>6</b>	<b>Autômatos Finitos Não Deterministas (AFND)</b>	<b>34</b>
6.1	Definição de autômato finito não determinista . . . . .	35
6.2	Árvore de caminhos . . . . .	36
6.3	Linguagem reconhecida por um AFND . . . . .	37
6.4	Fecho- $\epsilon$ . . . . .	38
<b>7</b>	<b>Equivalência entre AFD e AFND</b>	<b>39</b>
7.1	Conversão de AFND em AFD . . . . .	39

<b>8</b>	<b>Operações sobre AFD e AFND</b>	<b>43</b>
8.1	Reunião de autómatos finitos . . . . .	43
8.2	Concatenação de autómatos finitos . . . . .	46
8.3	Fecho de Kleene de autómatos finitos . . . . .	48
8.4	Complementação de autómatos finitos . . . . .	50
8.5	Intersecção de autómatos finitos . . . . .	51
8.6	Diferença de autómatos . . . . .	53
<b>9</b>	<b>Equivalência entre ER e AF</b>	<b>54</b>
9.1	Conversão de ER em AF . . . . .	54
9.1.1	Autómatos dos elementos primitivos . . . . .	55
9.1.2	Algoritmo de conversão . . . . .	55
9.2	Conversão de AF em ER . . . . .	57
9.2.1	Autômato finito generalizado . . . . .	57
9.2.2	Algoritmo de conversão . . . . .	58
<b>10</b>	<b>Equivalência entre GR e AF</b>	<b>62</b>
10.1	Conversão de AF em GR . . . . .	63
10.2	Conversão de GR em AF . . . . .	64

# Capítulo 1

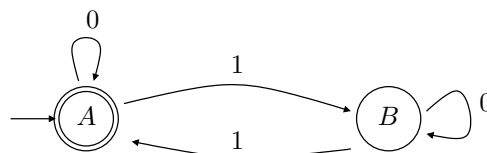
## Linguagens

Uma linguagem é um sistema de símbolos usado para comunicar informação. Uma mensagem nessa linguagem é uma sequência de símbolos, mas nem todas as sequências são válidas. Logo, uma linguagem é caracterizada por um conjunto de símbolos e uma forma de descrever as sequências de símbolos válidas, ou seja, uma linguagem é um conjunto de sequências definidas sobre um conjunto de símbolos.

### 1.1 Exemplos de linguagens

- O **conjunto dos vocábulos em português** é uma linguagem, cujos símbolos são as letras do alfabeto (incluindo as letras acentuadas e cedilhadas) e cujas sequências válidas são os vocábulos do português. É uma linguagem finita, pelo que as sequências válidas podem ser descritas por extenso. Poderão também ser descritas através de regras de produção. Por exemplo, as formas verbais de todos os verbos regulares terminados em ar são iguais, a menos de um radical. Estas palavras podem ser definidas pelo produto cartesiano dos possíveis radicais com as possíveis terminações.
- O **conjunto das sequências binárias com um número par de uns** é uma linguagem, cujos símbolos são os dígitos binários (0 e 1) e cujas sequências válidas são aquelas cujo número de uns é par.

Como representar as sequências válidas? O grafo seguinte é uma forma de o fazer.



Considerando que representa um caminho sobre o grafo, uma sequência pertence à linguagem se, começando no nó A, percorrer um caminho sobre o grafo e terminar no nó A.

- O **conjunto dos políndromos definidos com as letras a e b** é uma linguagem, cujos símbolos são as letras a e b e cujas sequências válidas são as que se leem da mesma maneira se lidas da esquerda para a direita ou vice-versa. Esta linguagem pode ser definida por indução:
  1. a, b, aa e bb são políndromos;
  2. se s é um políndromo, então também o são aSa e bSb.
- O **conjunto das sequências binárias começadas por '1' e terminadas em '0'** é uma linguagem que pode ser representada pela expressão (regular)  $1(0|1)^*0$ . (As expressões regulares são tratadas mais à frente.)
- O **conjunto das sequências reconhecidas por uma máquina de calcular** é uma linguagem, cujos símbolos são os dígitos, o separador decimal (ponto ou vírgula), os operadores, etc., e cujas sequências válidas são aquelas que representam expressões aritméticas válidas. Por exemplo:  $10+1$  é uma expressão válida, mas  $10++1$  não o é.
- A **língua portuguesa** é uma linguagem, cujos símbolos são os vocábulos do português (mais os sinais de pontuação), e cujas sequências são os textos em português. É um conjunto infinito, pelo que as sequências válidas apenas podem ser descritas através de produções gramaticais, a gramática do português.

## 1.2 Elementos básicos sobre linguagens

- O **símbolo**, também designado por **letra**, é o átomo do mundo das linguagens.
- O **alfabeto** é o conjunto de símbolos de suporte a uma dada linguagem. Deve ser um conjunto finito, não vazio, de símbolos. Exemplos:  $A_1 = \{0, 1, \dots, 9\}$  é o alfabeto do conjunto dos números inteiros positivos;  $A_2 = \{0, 1\}$  é o alfabeto do conjunto dos números binários.
- A **palavra**, também designada por *string*, é uma sequência de símbolos sobre um dado alfabeto,

$$u = a_1 a_2 \cdots a_n, \quad \text{com } a_i \in A \wedge n \geq 0$$

Exemplos:

- 00011 é uma palavra sobre o alfabeto  $\{0, 1\}$ ;
- abbbbcc é uma palavra sobre o alfabeto  $\{a, b, c\}$ .

- O **comprimento** de uma palavra  $u$  denota-se por  $|u|$  e representa o seu número de símbolos.

- É habitual interpretar-se a palavra  $u$  como uma função

$$u : \{1 \cdots n\} \rightarrow A, \quad \text{com } n = |u|$$

Se  $u = abc$ , então  $u_1 = a$ ,  $u_2 = b$  e  $u_3 = c$ .

- A **palavra vazia** é uma sequência de 0 (zero) símbolos e denota-se por  $\varepsilon$ .<sup>1</sup>

Note que  $\varepsilon$  não pertence ao alfabeto.

- $A^*$  representa o conjunto de todas as palavras sobre o alfabeto  $A$ , incluindo a palavra vazia. Por exemplo, se  $A = \{0, 1\}$ , então  $A^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$ .

Note que, dada um alfabeto  $A$  qualquer, qualquer **linguagem** sobre  $A$  mais não é do que um subconjunto de  $A^*$ . Note ainda que  $\{\}$ ,  $\{\varepsilon\}$  e  $A^*$  são subconjuntos de  $A$ , qualquer que seja o  $A$ , e por conseguinte são linguagens sobre o alfabeto  $A$ .

## 1.3 Operações sobre palavras

A **concatenação**, ou **produto**, das palavras  $u$  e  $v$  denota-se por  $u.v$ , ou simplesmente  $uv$ , e representa a justaposição de  $u$  e  $v$ , i.e., a palavra constituída pelos símbolos de  $u$  seguidos dos símbolos de  $v$ . Note que  $|u.v| = |u| + |v|$ .

- A concatenação goza da propriedade **associativa**:  $u.(v.w) = (u.v).w = u.v.w$ .
- A concatenação goza da propriedade **existência de elemento neutro**:  $u.\varepsilon = \varepsilon.u = u$ .

A **potência** de ordem  $n$ , com  $n \geq 0$ , de uma palavra  $u$  denota-se por  $u^n$  e representa a concatenação de  $n$  réplicas de  $u$ , ou seja,  $\underbrace{uu \cdots u}_{n \times}$ . Note que  $u^0$  é igual a  $\varepsilon$ , qualquer que seja o  $u$ .

**Prefixo** de uma palavra  $u$  é uma sequência com parte dos símbolos iniciais de  $u$ ; **sufixo** de uma palavra  $u$  é uma sequência com parte dos símbolos finais de  $u$ ; **sub-palavra** de uma palavra  $u$  é uma sequência de parte dos símbolos intermédios de  $u$ . Note que  $\varepsilon$  e  $u$  são prefixos, sufixos e sub-palavras de  $u$ , qualquer que seja o  $u$ .

O **reverso** de uma palavra  $u$  é a palavra, denotada por  $u^R$ , que se obtém invertendo a ordem dos símbolos de  $u$ , i.e., se  $u = u_1 u_2 \cdots u_n$  então  $u^R = u_n \cdots u_2 u_1$ .

Denota-se por  $\#(x, u)$  a função que devolve o número de ocorrências do símbolo  $x$  na palavra  $u$ .

<sup>1</sup>Nas linguagens de programação, é habitual representar-se as *strings* entre aspas. Nestes casos uma *string* vazia é representada por ""



## 1.4 Operações sobre linguagens

O conjunto das linguagens definíveis sobre um alfabeto  $A$  é fechado sobre as seguintes operações: reunião, intersecção, diferença, complementação, concatenação, potenciação e fecho de Kleene. Apresentam-se a seguir as definições destas operações assim como algumas propriedades de que gozam. As operações serão ilustradas com exemplos, tomando como ponto de partida as linguagens  $L_a$  e  $L_b$  definidas sobre o alfabeto  $A = \{a, b, c\}$  da seguinte maneira:

$$\begin{aligned} L_a &= \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\} \\ L_b &= \{u \mid u \text{ termina com } a\} = \{wa \mid w \in A^*\} \end{aligned}$$

### 1.4.1 Reunião

A **reunião** de duas linguagens  $L_1$  e  $L_2$  denota-se por  $L_1 \cup L_2$  e é definida da seguinte forma:

$$L_1 \cup L_2 = \{u \mid u \in L_1 \vee u \in L_2\} \quad (1.1)$$

#### Exemplo 1.1

$$\begin{aligned} \text{Sendo } L_a &= \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\} \text{ e } L_b = \{u \mid u \text{ termina com } a\} = \{wa \mid w \in A^*\} \\ L_a \cup L_b &= \{u \mid u \text{ começa por } a \vee u \text{ termina com } a\} \\ &= \{xwy \mid w \in A^* \wedge (x = a \vee y = a)\} \cup \{a\} \\ &= \{w_1aw_2 \mid w_1, w_2 \in A^* \wedge (w_1 = \varepsilon \vee w_2 = \varepsilon)\} \end{aligned}$$

### 1.4.2 Intersecção

A **intersecção** de duas linguagens  $L_1$  e  $L_2$  denota-se por  $L_1 \cap L_2$  e é definida da seguinte forma:

$$L_1 \cap L_2 = \{u \mid u \in L_1 \wedge u \in L_2\}$$

#### Exemplo 1.2

$$\begin{aligned} \text{Sendo } L_a &= \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\} \text{ e } L_b = \{u \mid u \text{ termina com } a\} = \{wa \mid w \in A^*\} \\ L_a \cap L_b &= \{u \mid u \text{ começa por } a \wedge u \text{ termina com } a\} \\ &= \{awa \mid w \in A^*\} \cup \{a\} \end{aligned}$$

### 1.4.3 Diferença e complementação

A **diferença** entre duas linguagens  $L_1$  e  $L_2$  denota-se por  $L_1 \setminus L_2$  e é definida da seguinte forma<sup>2</sup>:

$$L_1 \setminus L_2 = \{u \mid u \in L_1 \wedge u \notin L_2\}$$

#### Exemplo 1.3

Sendo  $L_a = \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\}$  e  $L_b = \{u \mid u \text{ termina com } a\} = \{wa \mid w \in A^*\}$

$$\begin{aligned} L_a \setminus L_b &= \{u \mid u \text{ começa por } a \wedge u \text{ não termina com } a\} \\ &= \{awx \mid w \in A^* \wedge x \in A \wedge x \neq a\} \end{aligned}$$

A linguagem **complementar** da linguagem  $L$  denota-se por  $\overline{L}$  e é definida da seguinte forma:

$$\overline{L} = \{u \mid u \in A^* \wedge u \notin L\}$$

#### Exemplo 1.4

Sendo  $L = \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\}$

$$\begin{aligned} \overline{L} &= A^* \setminus L = \{u \mid u \notin L\} \\ &= \{u \mid u \text{ não começa por } a\} = \{xw \mid w \in A^* \wedge x \in A \wedge x \neq a\} \cup \{\varepsilon\} \end{aligned}$$

### 1.4.4 Concatenação, potência e fecho de Kleene

A **concatenação** de duas linguagens  $L_1$  e  $L_2$  denota-se por  $L_1.L_2$  e é definida da seguinte forma:

$$L_1.L_2 = \{uv \mid u \in L_1 \wedge v \in L_2\}$$

#### Exemplo 1.5

Sendo  $L_a = \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\}$  e  $L_b = \{u \mid u \text{ termina com } a\} = \{wa \mid w \in A^*\}$

$$\begin{aligned} L_a.L_b &= \{uv \mid u \text{ começa por } a \wedge v \text{ termina com } a\} \\ &= \{awa \mid w \in A^*\} \end{aligned}$$

A **potência** de ordem  $n$  da linguagem  $L$  denota-se por  $L^n$  e é definida indutivamente da seguinte forma:

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^{n+1} &= L^n.L \end{aligned}$$

<sup>2</sup>Frequentemente, também é usada a notação  $L_1 - L_2$

**Exemplo 1.6**

Sendo  $L = \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\}$

$$L^0 = \{\varepsilon\}$$

$$L^1 = L^0.L = \{\varepsilon\}.L = L = \{aw \mid w \in A^*\}$$

$$L^2 = L^1.L = \{auav \mid u, v \in A^*\}$$

O **fecho de Kleene** da linguagem  $L$  denota-se por  $L^*$  e é definido da seguinte forma:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{i=0}^{\infty} L^i$$

**Exemplo 1.7**

Sendo  $L = \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\}$

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots$$

$$= L^0 \cup L^1$$

$$= L \cup \{\varepsilon\}$$

Para perceber a simplificação anterior note que  $L^{n+1} \subset L^n$ ,  $n > 0$ .

**1.4.5 Propriedades das várias operações**

Sendo as linguagens conjuntos, todas as propriedades aplicáveis a conjunto são naturalmente aplicáveis a linguagens. Assim, as operações de reunião e intersecção gozam das propriedades comutativa, associativa e distributiva.

As operações de reunião, intersecção e complementação gozam das **leis de De Morgan**:

$$L_1 - (L_2 \cup L_3) = (L_1 - L_2) \cap (L_1 - L_3)$$

$$L_1 - (L_2 \cap L_3) = (L_1 - L_2) \cup (L_1 - L_3)$$

A concatenação goza das propriedades:

**associativa:**  $L_1.(L_2.L_3) = (L_1.L_2).L_3 = L_1.L_2.L_3$

**existência de elemento neutro:**  $L.\{\varepsilon\} = \{\varepsilon\}.L = L$

**existência de elemento absorvente:**  $L.\emptyset = \emptyset.L = \emptyset$

**distributiva em relação à reunião:**  $L_1.(L_2 \cup L_3) = L_1.L_2 \cup L_1.L_3$

**distributiva em relação à intersecção:**  $L_1.(L_2 \cap L_3) = L_1.L_2 \cap L_1.L_3$

## Capítulo 2

# Linguagens Regulares (LR) e Expressões Regulares (ER)

### 2.1 Definição de linguagem regular

A classe das **linguagens regulares** sobre o alfabeto  $A$  define-se indutivamente da seguinte forma:

1. O conjunto vazio,  $\emptyset$ , é uma linguagem regular.
2. Qualquer que seja o  $a \in A$ , o conjunto  $\{a\}$ <sup>1</sup> é uma linguagem regular.
3. Se  $L_1$  e  $L_2$  são linguagens regulares, então  $L_1 \cup L_2$  é uma linguagem regular.
4. Se  $L_1$  e  $L_2$  são linguagens regulares, então  $L_1.L_2$  é uma linguagem regular.
5. Se  $L$  é uma linguagem regular, então  $L^*$  é uma linguagem regular.
6. Nada mais é linguagem regular.

Note que  $\{\varepsilon\}$  é uma linguagem regular, uma vez que  $\emptyset^* = \{\varepsilon\}$  e  $\emptyset$  é uma linguagem regular.

#### Exemplo 2.1

Mostre que o conjunto dos números binários começados em 1 e terminados em 0 é uma linguagem regular sobre o alfabeto  $A = \{0, 1\}$ .

**Resposta:** O conjunto pretendido pode ser representado por  $L = \{1\} \cdot A^* \cdot \{0\}$ . Este conjunto pode ser obtido indutivamente da seguinte forma:

1.  $\{0\}$  e  $\{1\}$  são regulares pela regra 2.
2.  $A = \{0, 1\} = \{0\} \cup \{1\}$  é regular por aplicação da regra 3.

<sup>1</sup>Note que o elemento do conjunto é uma palavra com uma letra e não uma letra. Se se fizesse uma analogia com linguagens de programação como o C/C++ ou o Java, o elemento do conjunto corresponde à palavra "a" e não à letra 'a'.

3. Se  $A$  é regular,  $A^*$  também o é por aplicação da regra 5.
4. Finalmente,  $\{1\} \cdot A^* \cdot \{0\}$  é regular por aplicação 2 vezes da regra 4.

### Exemplo 2.2

Mostre que o conjunto  $N = \{0, 1, 2, 3, \dots, 10, \dots\}$ , conjunto dos números inteiros positivos, é uma linguagem regular sobre o alfabeto  $A = \{0, 1, 2, \dots, 9\}$ .

**Resposta:** O conjunto pretendido corresponde ao conjunto formado pela palavra 0 mais todas as seqüência de 1 ou mais dígitos decimais, começadas por um dígito diferente de 0. Ou seja,  $N = \{0\} \cup A' \cdot A^*$  onde  $A' = \{1, 2, \dots, 9\}$ .  $N$  pode ser obtido indutivamente da seguinte forma:

1. Qualquer que seja o  $d \in A$ ,  $\{d\}$  é uma linguagem regular pela regra 2.
2.  $A = \{0, 1, 2, \dots, 9\}$  é uma linguagem regular, por aplicação sucessiva da regra 3.
3.  $A' = \{1, 2, \dots, 9\}$  é uma linguagem regular, por aplicação sucessiva da regra 3.
4. Se  $A$  é uma linguagem regular,  $A^*$  também o é, por aplicação da regra 5.
5. Se  $A'$  e  $A^*$  são linguagens regulares,  $A' \cdot A^*$ , é uma linguagem regular por aplicação da regra 4.
6. Finalmente,  $N = \{0\} \cup A' \cdot A^*$  é uma linguagem regular por aplicação da regra 3.

Existem diversos formalismos para descrever/representar linguagens regulares. Iremos tratar os seguintes: **expressões regulares**, **gramáticas regulares**, **gramáticas regulares generalizadas**, **autômatos finitos deterministas**, **autômatos finitos não deterministas** e **autômatos finitos generalizados**, assim como os procedimentos de conversão de uns nos outros.

## 2.2 Expressões regulares

O conjunto das **expressões regulares** sobre o alfabeto  $A$  define-se indutivamente da seguinte forma:

1.  $\emptyset$  é uma expressão regular que representa a linguagem regular  $\emptyset$ .
2. Qualquer que seja o  $a \in A$ ,  $a$  é uma expressão regular que representa a linguagem regular  $\{a\}$ .
3. Se  $e_1$  e  $e_2$  são expressões regulares representando respetivamente as linguagens regulares  $L_1$  e  $L_2$ , então  $(e_1|e_2)$  é uma expressão regular que representa a linguagem regular  $L_1 \cup L_2$ .
4. Se  $e_1$  e  $e_2$  são expressões regulares representando respetivamente as linguagens regulares  $L_1$  e  $L_2$ , então  $(e_1e_2)$  é uma expressão regular que representa a linguagem regular  $L_1.L_2$ .
5. Se  $e_1$  é uma expressão regular representando a linguagem regular  $L_1$ , então  $e_1^*$  é uma expressão regular que representa a linguagem regular  $(L_1)^*$ .
6. Nada mais é expressão regular.

Note que é habitual representar-se por  $\varepsilon$  a expressão regular  $\emptyset^*$ . Esta expressão representa a linguagem regular formada apenas pela palavra vazia,  $(\{\varepsilon\})$ .

### Exemplo 2.3

Obtenha uma expressão regular que represente o conjunto  $N$  do exemplo anterior (exemplo 2.2).

**Resposta:** Uma expressão regular que representa  $N$  é

$$e = 0|(((((((1|2|3|4|5|6|7|8|9)(((((((0|1|2|3|4|5|6|7|8|9)*)$$

Esta expressão tem a forma  $e_1|(e_2.e_3^*)$ , com  $e_1 = 0$ ,  $e_2 = (((((((((1|2|3|4|5|6|7|8|9)$  e  $e_3 = (((((((((0|1|2|3|4|5|6|7|8|9)$ . A expressão  $e_1$  representa o elemento 0. A expressão  $e_2.e_3^*$  representa as sequências começadas por um dígito diferente de 0. A expressão

$$e = 0|(1|2|\dots|9)(0|1|2|\dots|9)^*$$

representa o mesmo conjunto e é claramente mais legível que a anterior. No entanto, a sua utilização só é válida por causa de propriedades de que gozam os operadores das expressões regulares e que serão apresentados a seguir.

## 2.3 Propriedades das expressões regulares

A operação de escolha ( $|$ ) goza das propriedades **associativa**, **comutativa**, **existência de elemento neutro** e **idempotência**.

1. As propriedades **associativa** e **comutativa** estabelecem respetivamente que

$$e_1|(e_2|e_3) = (e_1|e_2)|e_3 = e_1|e_2|e_3$$

e que

$$e_1|e_2 = e_2|e_1$$

2. A expressão vazia, representando o conjunto vazio, é o **elemento neutro** da operação de escolha

$$e_1|\emptyset = \emptyset|e_1 = e_1$$

3. A operação de escolha goza ainda de **idempotência**

$$e_1|e_1 = e_1$$

A operação de concatenação goza das propriedades **associativa**, **existência de elemento neutro** e **existência de elemento absorvente**.

1. A propriedade **associativa** estabelece que

$$e_1(e_2e_3) = (e_1e_2)e_3 = e_1e_2e_3$$

2. A palavra vazia, representando o conjunto formado simplesmente pela palavra vazia ( $\{\varepsilon\}$ ) é o **elemento neutro** da concatenação

$$e_1\varepsilon = \varepsilon e_1 = e_1$$

3. A expressão vazia, representando o conjunto vazio, é o **elemento absorvente** da concatenação

$$e_1\emptyset = \emptyset e_1 = \emptyset$$

Finalmente, os operadores de escolha e concatenação gozam das propriedades **distributiva à esquerda da concatenação em relação à escolha**

$$e_1(e_2|e_3) = e_1e_2|e_1e_3$$

e **distributiva à direita da concatenação em relação à escolha**

$$(e_1|e_2)e_3 = e_1e_3|e_2e_3$$

Note que o fecho de Kleene não goza da propriedade distributiva. Quer isto dizer que em geral

$$(e_1|e_2)^* \neq e_1^*|e_2^*$$

$$e_1 \neq \emptyset \wedge e_2 \neq \emptyset \wedge e_1 \neq \emptyset \wedge e_2 \neq \emptyset$$

e

$$(e_1e_2)^* \neq e_1^*e_2^*$$

## 2.4 Simplificação notacional

Na escrita de expressões regulares assume-se que a operação de fecho (\*) tem precedência em relação à operação de concatenação e esta tem precedência em relação à operação de escolha (|). O uso destas precedências, aliado às propriedades apresentadas, permite a queda de alguns parêntesis e consequentemente uma notação simplificada. A expressão final do exemplo 2.3 é um exemplo desta notação simplificada.

### Exemplo 2.4

Determine uma expressão regular que represente o conjunto das sequências binárias em que o número de 0 (zeros) é igual a 2.

**Resposta:** Não fazendo uso da notação simplificada apresentada atrás, obter-se-ia, por exemplo, a expressão regular

$$e = (((1^*)0)(1^*))0(1^*)$$

A aplicação das simplificações resulta em

$$\underline{1^*01^*01^*}$$

Não esquecer!

**Exemplo 2.5**

Sobre o alfabeto  $A = \{a, b, c\}$  construa uma expressão regular que reconheça a linguagem  $L$ , onde  $\#(a, w)$  é uma função que devolve o número de ocorrências da letra  $a$  na palavra  $w$ ,

$$L = \{w \in A^* \mid \#(a, w) = 3\}$$

**Resposta:** A expressão regular pretendida é

$$e = (b|c)^*a(b|c)^*a(b|c)^*a(b|c)^* \quad \text{! Segue sempre a mesma lógica!}$$

**2.5 Extensão notacional**

Mesmo com as simplificações notacionais apresentadas, por vezes, uma expressão regular pode ser difícil de entender. Isto é particularmente verdade quando os alfabetos de entrada são extensos.

**Exemplo 2.6**

Repita o exemplo 2.5 considerando que o alfabeto de entrada é o conjunto das letras minúsculas.

**Resposta:** Para construir a expressão regular pretendida basta substituir na resposta do exemplo 2.5 cada ocorrência de  $(b|c)$  por  $(b|c|d|\dots|y|z)$ , obtendo-se

$$e = e_1^* = ((b|c|d|\dots|y|z)^*a(b|c|d|\dots|y|z)^*a(b|c|d|\dots|y|z)^*a(b|c|d|\dots|y|z)^*)^*$$

Embora com um grau de complexidade igual ao do exemplo 2.5 esta expressão regular é mais difícil de ler.

Considere-se um exemplo onde esta dificuldade de leitura é ainda mais patente.

**Exemplo 2.7**

Considerando que o alfabeto de entrada é o conjunto das letras maiúsculas e minúsculas, construa uma expressão regular que represente o conjunto das palavras com um número par de vogais.

**Resposta:** A expressão regular pretendida é dada por

$$((b|c|d|f|\dots|y|z|B|C|D|F|\dots|Y|Z) \\ |(a|e|i|o|u|A|E|I|O|U)(b|c|d|f|\dots|y|z|B|C|D|F|\dots|Y|Z) * (a|e|i|o|u|A|E|I|O|U))^*$$

Torna-se, por isso, necessário definir métodos de representação das expressões regulares que possuam maior poder expressivo. Um dos métodos usados faz uso de várias extensões notacionais. Ir-se-ão considerar as seguintes extensões notacionais:



1. O operador sufixo  $+$  é usado para representar 1 ou mais ocorrências da expressão regular  $a$  que se aplica. Assim,

$$e+ \equiv ee^*$$

2. O operador sufixo  $?$  é usado para representar zero ou uma ocorrência da expressão  $a$  que se aplica. Assim,

$$e? \equiv (e|\varepsilon)$$

3. O símbolo  $.$  é usado para representar um símbolo qualquer do alfabeto<sup>2</sup>. Por exemplo, sobre o alfabeto das letras minúsculas,

$$. \equiv (a|b|c|\dots|y|z)$$

4. A expressão  $[x_1x_2x_3\dots x_n]$  é usada para representar um (note bem, **um e um só**) símbolo do conjunto  $\{x_1, x_2, x_3, \dots, x_n\}$ . Por exemplo, a expressão regular  $[aeiouAEIOU]$  representa uma vogal, maiúscula ou minúscula.

A construção anterior permite ainda especificar gamas de símbolos usando um intervalo de valores. A expressão  $[x_i - x_f]$  representa um símbolo do alfabeto entre  $x_i$  e  $x_f$ . Por exemplo,  $[a-z]$  representa uma letra minúscula.

É possível construir expressões juntando gamas e símbolos considerados individualmente. Por exemplo, a expressão  $[a-zA-Z0-9\_]$  representa uma letra, maiúscula ou minúscula, um algarismo decimal ou o símbolo ‘ $\_$ ’, ou seja, representa um símbolo do conjunto  $\{a, b, \dots, y, z, A, B, \dots, Y, Z, 0, 1, \dots, 8, 9, \_ \}$ .

5. A expressão  $[\hat{x}_1x_2x_3\dots x_n]$  pode ser usada para representar um símbolo do conjunto complementar do conjunto  $\{x_1, x_2, x_3, \dots, x_n\}$ . Por exemplo, sobre o alfabeto das letras minúsculas e maiúsculas a expressão  $[\hat{aeiouAEIOU}]$  representa uma consoante<sup>3</sup>.
6. A expressão  $e\{n\}$  representa  $n$  concatenações da expressão  $e$ , ou seja,  $e^n$ .
7. A expressão  $e\{n_1, n_2\}$  representa entre  $n_1$  e  $n_2$  concatenações da expressão  $e$ , ou seja, a expressão  $e^{n_1}|e^{n_1+1}|\dots|e^{n_2}$ .
8. A expressão  $e\{n, \}$  representa  $n$  ou mais concatenações da expressão  $e$  a expressão  $e^n|e^{n+1}|\dots$ .

Note, no entanto, que as extensões notacionais disponíveis podem variar dependendo da ferramenta usada<sup>4</sup>.

<sup>2</sup>Algumas ferramentas, como a linguagem lexical `flex`, excluem do  $.$  a mudança de linha.

<sup>3</sup>Note que o que é dito apenas é verdade sobre o alfabeto das letras. A mesma expressão terá um significado diferente se o alfabeto for outro. Por exemplo, nas linguagens que aceitam expressões regulares, o alfabeto é todo o código ASCII. Nesses casos, a expressão representaria qualquer símbolo à exceção dos das vogais.

<sup>4</sup>Por exemplo, em *ANTLR*, as extensões 5 a 8 não existem nessa forma. A 5 aparece na forma  $\sim[\dots]$ . As outras aparecem na forma de predicados semânticos.

## 2.6 Aplicação das expressão regulares

- **Validação** de entrada, por exemplo em formulários.
- **Procura e seleção** de texto. Ferramentas como o `word` e o `libre office` têm suporte de expressões regulares.
- **Tokenization**, que corresponde à transformação de uma sequência de caracteres numa sequência de *tokens*, parte do processo de compilação.

## 2.7 Exercícios

### Exercício 2.1

Determine uma expressão regular que represente as sequências binárias com um número par de uns.

$0^*(10^*10^*)^*$

### Exercício 2.2

Determine uma expressão regular que represente as constantes numéricas na linguagem C. Eis alguns exemplos de constantes numéricas em C: “10”, “10.1”, “10.”, “.12”, “1e5”, “10.1E+4”.

## Capítulo 3

# Gramáticas regulares (GR)

Considere uma estrutura  $G$ , definido sobre o alfabeto  $T = \{a, b\}$ , com as regras de rescrita

$$\begin{aligned} S &\rightarrow b \\ S &\rightarrow a S \end{aligned}$$

significando que em qualquer sequência onde apareça o símbolos  $S$  ele pode ser substituído (rescrito) por  $b$  ou por  $a S$ . É habitual usar-se o símbolo ‘|’ para denotar as várias alternativas de rescrita associadas ao mesmo símbolo. Assim, a estrutura anterior é comumente representada por

$$\begin{aligned} S &\rightarrow b \\ &| a S \end{aligned}$$

Que palavras apenas constituídas por símbolos do alfabeto  $T$  se podem gerar a partir de  $S$ ? Substituindo  $S$  por  $b$  obtém-se  $b$  que pertence a  $T^*$ . Isto pode representar-se por

$$S \Rightarrow b$$

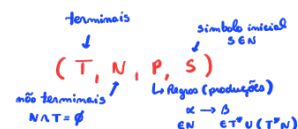
Substituindo  $S$  por  $a S$  e o novo  $S$  por  $b$ , obtém-se  $ab$ , escrevendo-se

$$S \Rightarrow a S \Rightarrow a b$$

Na realidade, começando em  $S$  podem gerar-se as palavras  $b$ ,  $ab$ ,  $aab$ ,  $aa \cdots ab$ , ou seja, todas as palavras da linguagem  $L$  dada por

$$L = \{a^n b \mid n \geq 0\}$$

ou, equivalentemente, as palavras descritas pela expressão regular  $a^*b$ .



A estrutura  $G$  é uma **gramática** que permite gerar a linguagem  $L$ .  $T$  é o alfabeto de entrada, também designado por conjunto de símbolos terminais, ou simplesmente conjunto de terminais.  $S$  é um símbolo não-terminal, ponto de partida de todas as palavras. É designado **símbolo inicial da gramática**. As regras de rescrita são designadas **produções**. O conjunto das palavras que se podem gerar a partir do símbolo inicial da gramática por aplicação sucessiva de produções é a linguagem descrita por  $G$ .

As gramáticas podem pertencer a diferentes categorias, dependendo da definição das suas produções. Para já, apenas nos interessa abordar uma dessas categorias, as **gramáticas regulares**.

### 3.1 Definição de gramática regular

Formalmente, uma gramática regular é um quádruplo  $G = (T, N, P, S)$ , onde

- $T$  é um conjunto finito não vazio de símbolos terminais;
- $N$ , sendo  $N \cap T = \emptyset$ , é um conjunto finito não vazio de símbolos não terminais;
- $P$  é um conjunto de produções, cada uma da forma  $\alpha \rightarrow \beta$ , onde
  - $\alpha \in N$
  - $\beta \in T^* \cup (T^*N)$
- $S \in N$  é o símbolo inicial.

Nas produções,  $\alpha$  e  $\beta$  são designados por cabeça da produção e corpo da produção, respetivamente.

▽ Note que os corpos das produções numa gramática regular ou apenas têm símbolos terminais (zero ou mais) ou têm apenas um símbolo não terminal, necessariamente no fim.

### 3.2 Operações sobre gramáticas regulares

As gramáticas regulares são fechadas sob as operações de reunião, concatenação, fecho, intersecção e complementação.



#### 3.2.1 Reunião de gramáticas regulares

Sejam  $G_1 = (T_1, N_1, P_1, S_1)$  e  $G_2 = (T_2, N_2, P_2, S_2)$  duas gramáticas regulares quaisquer, com  $N_1 \cap N_2 = \emptyset$ . A gramática  $G = (T, N, P, S)$  onde

$$\begin{aligned} T &= T_1 \cup T_2 \\ N &= N_1 \cup N_2 \cup \{S\} \quad \text{com } S \notin (N_1 \cup N_2) \\ P &= \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2 \end{aligned}$$

→ "Adicionar" à união das produções o símbolo inicial!

é regular e gera a linguagem  $L = L(G_1) \cup L(G_2)$ . As novas produções,  $S \rightarrow S_1$  e  $S \rightarrow S_2$ , permitem gerar, respetivamente, as palavras da primeira e segunda gramáticas, contribuindo dessa forma para a reunião.

#### Exemplo 3.1

Sobre o conjunto de terminais  $T = \{a, b, c\}$ , determine uma gramática regular que represente a linguagem  $L = L_1 \cup L_2$  sabendo que

$$L_1 = \{\omega a : \omega \in T^*\} \quad (\text{palavras terminadas em } a)$$

e

$$L_2 = \{a\omega : \omega \in T^*\} \quad (\text{palavras começadas por } a)$$

**Resposta:**Comece-se por determinar as gramáticas para as linguagens  $L_1$  e  $L_2$ 

$$\begin{array}{ll} S_1 \rightarrow a S_1 & S_2 \rightarrow a X_2 \\ | b S_1 & X_2 \rightarrow a X_2 \\ | c S_1 & | b X_2 \\ | a & | c X_2 \\ & | \varepsilon \end{array}$$

A aplicação do algoritmo da reunião resulta em

$$\begin{array}{lll} S \rightarrow S_1 & S_1 \rightarrow a S_1 & S_2 \rightarrow a X_2 \\ | S_2 & | b S_1 & X_2 \rightarrow a X_2 \\ & | c S_1 & | b X_2 \\ & | a & | c X_2 \\ & & | \varepsilon \end{array}$$

### 3.2.2 Concatenação de gramáticas regulares

Sejam  $G_1 = (T_1, N_1, P_1, S_1)$  e  $G_2 = (T_2, N_2, P_2, S_2)$  duas gramáticas regulares quaisquer, com  $N_1 \cap N_2 = \emptyset$ . A gramática  $G = (T, N, P, S)$  onde

$$\begin{aligned} T &= T_1 \cup T_2 \\ N &= N_1 \cup N_2 \cup \{S\} \quad \text{com } S \notin (N_1 \cup N_2) \\ P &= \{A \rightarrow \omega S_2 : (A \rightarrow \omega) \in P_1 \wedge \omega \in T_1^*\} \\ &\quad \cup \{A \rightarrow \omega : (A \rightarrow \omega) \in P_1 \wedge \omega \in T_1^* N_1\} \\ &\quad \cup P_2 \\ S &= S_1 \end{aligned}$$

é regular e gera a linguagem  $L = L(G_1).L(G_2)$ , concatenação de  $L(G_1)$  com  $L(G_2)$ . As produções de  $G_1$  apenas constituídas por símbolos terminais transitam para  $G$  sendo-lhes acrescentado no fim o símbolo inicial de  $G_2$  ( $S_2$ ). As restantes produções de  $G_1$  (aquelas que terminal num símbolo não terminal) e todas as produções de  $G_2$  transitam inalteradas. O símbolo inicial da concatenação corresponde ao (ou transforma-se diretamente no) símbolo inicial de  $G_1$ .

**Exemplo 3.2**

Sobre o conjunto de terminais  $T = \{a, b, c\}$ , determine uma gramática regular que represente a linguagem  $L = L_1.L_2$  sabendo que

$$L_1 = \{\omega a : \omega \in T^*\} \quad (\text{palavras terminadas em } a)$$

e

$$L_2 = \{a\omega : \omega \in T^*\} \quad (\text{palavras começadas por } a)$$

**Resposta:**

As linguagens  $L_1$  e  $L_2$  são as mesmas do exemplo anterior. Apresentam-se novamente aqui as suas gramáticas por conveniência.

$$\begin{array}{ll} S_1 \rightarrow a S_1 & S_2 \rightarrow a X_2 \\ | b S_1 & X_2 \rightarrow a X_2 \\ | c S_1 & | b X_2 \\ | a & | c X_2 \\ & | \varepsilon \end{array}$$

A aplicação do algoritmo da concatenação resulta em

$$\begin{array}{lll} S \rightarrow S_1 & S_1 \rightarrow a S_1 & S_2 \rightarrow a X_2 \\ & | b S_1 & X_2 \rightarrow a X_2 \\ & | c S_1 & | b X_2 \\ & | a S_2 & | c X_2 \\ & & | \varepsilon \end{array}$$

A única produção que transitou alterada foi a  $S_1 \rightarrow a$ , que passou a  $S_1 \rightarrow a S_2$ .

**3.2.3 Fecho de Kleene de gramáticas regulares**

Seja  $G_1 = (T_1, N_1, P_1, S_1)$  uma gramática regular qualquer. A gramática  $G = (T, N, P, S)$  onde

$$\begin{aligned} T &= T_1 \\ N &= N_1 \cup \{S\} \quad \text{com } S \notin N_1 \\ P &= \{S \rightarrow \varepsilon, S \rightarrow S_1\} \\ &\quad \cup \{A \rightarrow \omega S : (A \rightarrow \omega) \in P_1 \wedge \omega \in T_1^*\} \\ &\quad \cup \{A \rightarrow \omega : (A \rightarrow \omega) \in P_1 \wedge \omega \in T_1^* N_1\} \end{aligned}$$

é regular e gera a linguagem  $L = (L(G_1))^*$ . O fecho corresponde a concatenações da linguagem com ela própria. Assim, as produções de  $G_1$  apenas constituídas por símbolos terminais ganham o novo símbolo inicial ( $S$ ) no fim. As restantes produções de  $G_1$  (aquelas que terminam num símbolo não terminal) mantêm-se inalteradas. O novo símbolo inicial define o ponto de fuga do processo recursivo.

**Exemplo 3.3**

Sobre o conjunto de terminais  $T = \{a, b, c\}$ , determine uma gramática regular que represente a linguagem  $L = L_1^*$ , sabendo que

$$L_1 = \{\omega a : \omega \in T^*\} \quad (\text{palavras terminadas em } a)$$

**Resposta:** A linguagem  $L_1$  já foi abordada anteriormente. Apresenta-se novamente aqui a sua gramática por conveniência.

$$\begin{array}{lcl} S_1 & \rightarrow & a S_1 \\ & | & b S_1 \\ & | & c S_1 \\ & | & a \end{array}$$

A aplicação do algoritmo de fecho resulta em

$$\begin{array}{lcl} S & \rightarrow & \varepsilon \\ & | & S_1 \end{array} \quad \begin{array}{lcl} S_1 & \rightarrow & a S_1 \\ & | & b S_1 \\ & | & c S_1 \\ & | & a S \end{array}$$

*Não esquecer que o fecho inclui o vazio ( $\varepsilon$ )*

**3.3 Definição de gramática regular generalizada**

Para suporte aos procedimentos de conversão de gramáticas regulares em expressões regulares, apresentados noutro capítulo, mas não só, é possível generalizar a definição de gramática regular.

Formalmente, uma gramática regular generalizada é um quádruplo  $G = (T, N, P, S)$ , onde

- $T$  é um conjunto finito não vazio de símbolos terminais;
- $N$ , sendo  $N \cap T = \emptyset$ , é um conjunto finito não vazio de símbolos não terminais;
- $P$  é um conjunto de produções, cada uma da forma  $\alpha \rightarrow \beta$ , onde
  - $\alpha \in N$
  - $\beta \in E(T) \cup E(T)N$  ⚠
- $S \in N$  é o símbolo inicial.

onde  $E(T)$  representa o conjunto das expressões regulares definidas sobre o alfabeto  $T$ .

Note que esta definição abrange a dada anteriormente para gramática regular, uma vez que  $T^*$  representa um subconjunto das expressões regular sobre o alfabeto  $T$ .

## Capítulo 4

# Equivalência entre Expressões Regulares e Gramáticas Regulares

As expressões regulares e as gramáticas regulares são equivalentes, no sentido em que descrevem a mesma classe de linguagens, as linguagens regulares. Assim sendo, é possível converter uma expressão regular dada numa gramática regular que represente a mesma linguagem. Inversamente, é também possível converter uma gramática regular dada numa expressão regular descrevendo a mesma linguagem.

### 4.1 Conversão de uma expressão regular numa gramática regular

Como vimos na sua definição, uma expressão regular é contruída à custa de elementos primitivos e dos operadores escolha ( $|$ ), concatenação ( $.$ ) e fecho ( $*$ ). Os elementos primitivos correspondem à expressão  $\emptyset$ , representando o conjunto vazio, e às expressões do tipo  $a$  com  $a \in A$ , sendo  $A$  o alfabeto. É habitual considerar a expressão  $\varepsilon$  como um elemento primitivo, embora se saiba que  $\varepsilon = \emptyset^*$ .

É possível decompor-se uma expressão regular num conjunto de elementos primitivos interligados pelos operadores referidos acima. Dada uma expressão regular qualquer ela é:

1. um elemento primitivo;
2. ou uma expressão do tipo  $e^*$ , sendo  $e$  uma expressão regular qualquer;
3. ou uma expressão do tipo  $e_1.e_2$ , sendo  $e_1$  e  $e_2$  duas expressões regulares quaisquer;
4. ou uma expressão do tipo  $e_1|e_2$ , sendo  $e_1$  e  $e_2$  duas expressões regulares quaisquer;



Se se identificar as gramáticas regulares equivalentes das expressões primitivas, tem-se o problema da conversão de uma expressão regular para uma gramática regular resolvido, visto que se sabe como fazer a reunião, concatenação e fecho de gramáticas regulares (ver secção 3.2).

#### 4.1.1 Gramáticas regulares dos elementos primitivos

A tabela seguinte mostra as gramáticas regulares correspondentes às expressões regulares  $\varepsilon$  e  $a$ , sendo  $a$  um símbolo qualquer do alfabeto.

expressão regular	gramática regular
$\varepsilon$	$S \rightarrow \varepsilon$
$a$	$S \rightarrow a$

#### 4.1.2 Algoritmo de conversão

A transformação de uma expressão regular numa gramática regular pode ser feita aplicando os seguintes passos:

1. Se a expressão regular é o do tipo primitivo, a gramática regular correspondente pode ser obtida da tabela anterior.
2. Finalmente, se é do tipo  $e_1|e_2$ , aplica-se este mesmo algoritmo na obtenção de gramáticas regulares para as expressões  $e_1$  e  $e_2$  e, de seguida, aplica-se a reunião de gramáticas regulares descrita na secção 3.2.1.
3. Se é do tipo  $e_1.e_2$ , aplica-se este mesmo algoritmo na obtenção de gramáticas regulares para as expressões  $e_1$  e  $e_2$  e, de seguida, aplica-se a concatenação de gramáticas regulares descrita na secção 3.2.2.
4. Se é do tipo  $e^*$ , aplica-se este mesmo algoritmo na obtenção de uma gramática regular equivalente à expressão regular  $e$  e, de seguida, aplica-se o fecho de gramáticas regulares descrito na secção 3.2.3.

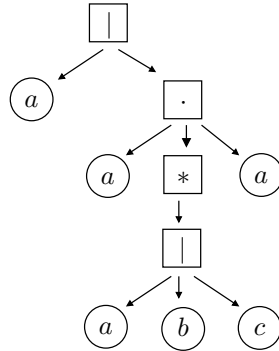
#### Exemplo 4.1

Construa uma gramática regular equivalente à expressão regular  $e = a|a(a|b|c)^*a$ .

##### Resposta:

A expressão regular  $e$  é do tipo  $e_1|e_2$ , com  $e_1 = a$  e  $e_2 = a(a|b|c)^*a$ , pelo que se deve aplicar a regra 2 do algoritmo. A expressão  $e_1$  é do tipo primitivo, pelo que se pode aplicar a tabela para obter a gramática regular correspondente. A expressão  $e_2$  resulta da concatenação de 3 expressões regulares, para as quais temos de obter gramáticas regulares. Apenas a segunda expressão da concatenação

tem de ser considerada, visto já se ter as gramáticas regulares das outras. Tem-se então que converter a expressão regular  $e_3 = (a|b|c)^*$ , que se obtém do fecho sobre a gramática regular que represente a expressão  $e_4 = a|b|c$ . Esta, por sua vez, resulta da reunião das gramáticas regulares das expressões primitivas  $a$ ,  $b$  e  $c$ . Esta decomposição pode ser visualizada na figura seguinte



Uma vez definida a decomposição, a construção faz-se dos elementos primitivos para a expressão global. As gramáticas regulares para as expressões  $a$ ,  $b$  e  $c$  são

$$S_a \rightarrow a \quad S_b \rightarrow b \quad S_c \rightarrow c$$

Fazendo a sua reunião obtém-se a gramática regular para  $e_4$

$$S_4 \rightarrow S_a \mid S_b \mid S_c$$

$$S_a \rightarrow a$$

$$S_b \rightarrow b$$

$$S_c \rightarrow c$$

que pode ser simplificada para

$$S_4 \rightarrow a \mid b \mid c$$

Aplicando o fecho a esta gramática obtém-se a gramática regular para a expressão  $e_3$

$$S_3 \rightarrow \varepsilon \mid S_4$$

$$S_4 \rightarrow a S_3 \mid b S_3 \mid c S_3$$

que pode ser simplificada para

$$S_3 \rightarrow \varepsilon \mid a S_3 \mid b S_3 \mid c S_3$$

A gramática regular para  $e_2$  resulta da concatenação das gramáticas regulares para  $a$ ,  $e_3$  e  $a$ , obtendo-se, após alguma simplificação, a gramática regular seguinte

$$S_2 \rightarrow a S_3$$

$$S_3 \rightarrow a \mid a S_3 \mid b S_3 \mid c S_3$$

Finalmente, a gramática regular para  $e$  obtém-se da reunião das gramáticas regulares para  $e_2$  e  $a$ , o que resulta na gramática regular seguinte

$$S \rightarrow a \mid S_2$$

$$S_2 \rightarrow a S_3$$

$$S_3 \rightarrow a \mid a S_3 \mid b S_3 \mid c S_3$$

que pode ser transformada para

$$S \rightarrow a \mid a S_3$$

$$S_3 \rightarrow a \mid a S_3 \mid b S_3 \mid c S_3$$

## 4.2 Conversão de uma gramática regular numa expressão regular

A conversão de gramáticas regulares em expressões regulares é feita através de gramáticas regulares generalizadas, com uma pequena alteração na forma como são usadas. Cada produção da forma  $X \rightarrow \beta$ , com  $\beta \in T^*$ , é representada pelo triplo  $(X, \beta, \varepsilon)$ . Por outro lado, cada produção da forma  $X \rightarrow \beta Y$ , com  $\beta \in T^*$  e  $Y \in N$ , é representada pelo triplo  $(X, \beta, Y)$ .

Convertendo inicialmente uma gramática regular para este formato e aplicando sucessivamente transformações de equivalência, o conjunto de triplos pode ser reduzido à forma  $(E, e, \varepsilon)$ , sendo  $e$  a expressão regular equivalente à gramática ponto de partida.

### 4.2.1 Algoritmo de conversão

Assim sendo, a obtenção de uma expressão regular equivalente a uma gramática regular qualquer dada, pode ser feita através dos seguintes passos:


1. Conversão de uma gramática  $(T, N, P, S)$  no conjunto de triplo seguintes

$$\begin{aligned} \mathcal{E} &= \{(E, \varepsilon, S)\} \\ &\cup \{(A, \omega, B) : (A \rightarrow \omega B) \in P \wedge B \in N\} \\ &\cup \{(A, \omega, \varepsilon) : (A \rightarrow \omega) \in P \wedge \omega \in T^*\} \end{aligned}$$

O triplo  $(E, \varepsilon, S)$  corresponde ao acrescento à gramática de uma nova produção  $E \rightarrow \varepsilon S$ , de forma a garantir que o (novo) símbolo inicial não aparece no corpo de nenhuma produção. Por conseguinte,  $E \notin N$ .

2. Eliminação, um a um, por transformações de equivalência, de todos os símbolos de  $N$ , até se obter um único triplo da forma  $(E, e, \varepsilon)$ .

A eliminação dos estados intermédios pode ser realizada pelo procedimento seguinte:

1. Substituição de todos os triplos da forma  $(A, \alpha_i, A)$ , com  $A \in N$ , por um único  $(A, \omega_2, A)$ , onde  $\omega_2 = \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m$
2. Substituição de todos os triplos da forma  $(A, \beta_i, B)$ , com  $A, B \in N$ , por um único  $(A, \omega_1, B)$ , onde  $\omega_1 = \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$
3. Substituição de cada triplo de triplos da forma  $(A, \omega_1, B), (B, \omega_2, B), (B, \omega_3, C)$ , com  $A, B, C \in N$ , pelo triplo  $(A, \omega_1 \omega_2^* \omega_3, C)$  
4. Repetição dos passos anteriores enquanto houver símbolos intermédios

**Exemplo 4.2**

Obtenha uma expressão regular equivalente à gramática regular seguinte

$$S \rightarrow a S \mid c S \mid aba X$$

$$X \rightarrow a X \mid c X \mid \varepsilon$$

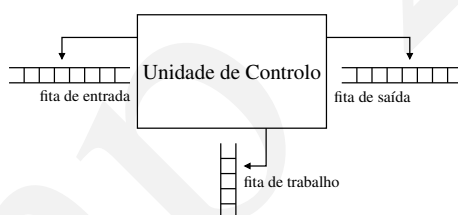
**Resposta:**

- Começa-se por converter a gramática num conjunto de triplos, acrescentando o triplo inicial  
 $\{ (E, \varepsilon, S), (S, a, S), (S, c, S), (S, aba, X), (X, a, X), (X, c, X), (X, \varepsilon, \varepsilon) \}$
- Substitui-se  $(S, a, S)$  e  $(S, c, S)$  por  $(S, (a|c), S)$   
 $\{ (E, \varepsilon, S), (S, (a|c), S), (S, aba, X), (X, a, X), (X, c, X), (X, \varepsilon, \varepsilon) \}$
- Substitui-se  $(X, a, X)$  e  $(X, c, X)$  por  $(X, (a|c), X)$   
 $\{ (E, \varepsilon, S), (S, (a|c), S), (S, aba, X), (X, (a|c), X), (X, \varepsilon, \varepsilon) \}$
- Substitui-se  $(E, \varepsilon, S)$ ,  $(S, (a|c), S)$  e  $(S, aba, X)$  por  $(E, (a|c)^* aba, X)$   
 $\{ (E, (a|c)^* aba, X), (X, (a|c), X), (X, \varepsilon, \varepsilon) \}$
- Finalmente, substitui-se  $(E, (a|c)^* aba, X)$ ,  $(X, (a|c), X)$  e  $(X, \varepsilon, \varepsilon)$  por  $(a|c)^* aba(a|c)^*$   
 $\{ (E, ((a|c)^* aba(a|c)^*), \varepsilon) \}$
- Obtendo-se a expressão regular pretendida  
 $(a|c)^* aba(a|c)^*$

## Capítulo 5

# Autómatos Finitos Deterministas (AFD)

Um *autômato finito* é um mecanismo reconhecedor das palavras de uma linguagem. Dada uma linguagem  $L$ , definida sobre um alfabeto  $A$ , um autômato finito reconhecedor de  $L$  é um mecanismo que reconhece as palavras de  $A^*$  que pertencem a  $L$ . Genericamente um autômato finito tem a configuração representada na figura seguinte.



Uma unidade de controlo, com capacidade finita de memorização, manipula os símbolos recolhidos de uma fita de entrada, que armazena uma palavra, e produz uma resposta. Definem-se vários tipos de autómatos, dependendo da forma como a fita de entrada é acedida e da existência ou inexistência de fitas de trabalho e de fita de saída.

A fita de entrada é uma unidade só de leitura com acesso sequencial ou aleatório aos símbolos da palavra. No acesso sequencial um símbolo da palavra de entrada lido e processado não pode ser processado novamente. Se assumirmos que a fita de entrada possui uma cabeça de leitura, esta apenas pode avançar posição a posição. No acesso aleatório o mesmo símbolo pode ser lido mais que uma vez, ou seja, assume-se que a cabeça de leitura pode ser deslocada para a frente ou para trás livremente.

Em geral, a resposta dos autómatos é do tipo sim/não ou aceite/rejeito. Quer isto dizer que um autômato não possui propriamente uma fita de saída, limitando-se a produzir um aceito se  $u \in L$  e um rejeito caso contrário. Neste sentido funcionam efetivamente como *reconhecedores* das palavras de uma linguagem. No entanto, há autómatos que produzem um símbolo de saída por cada símbolo de entrada processado. Nestes casos existe uma fita de saída que representa uma unidade só de escrita com acesso sequencial. Um símbolo de saída uma vez escrito não poderá ser apagado ou rescrito.

*Reconhecedor de palavras da linguagem*

Há autómatos em que a unidade de controlo recorre a fitas de trabalho para armazenar informação que auxilie no processamento. Estas fitas têm um funcionamento tipo pilha, ou seja, a ordem de colocação de elementos na fita é contrária à de retirada.

No resto deste capítulo ir-se-ão estudar categorias de autómatos caracterizadas pelo facto de possuírem uma fita de entrada com acesso sequencial e por não possuírem fitas de trabalho. Mais especificamente, ir-se-ão cobrir os autómatos finitos deterministas, os autómatos finitos não-deterministas e os autómatos finitos generalizados, que apenas produzem uma saída do tipo *aceito/rejeito*. Todos eles correspondem a modelos computacionais que representam linguagens regulares.

## 5.1 Definição de autómato finito determinista

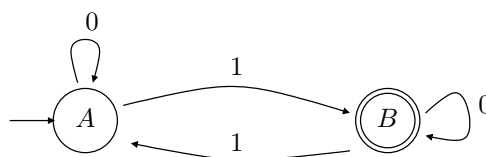
Um **autómato finito determinista** (AFD) é um quintuplo  $M = (A, Q, q_0, \delta, F)$ , em que:

- $A$  é o alfabeto de entrada;
- $Q$  é um conjunto finito não vazio de estados;
- $q_0 \in Q$  é o estado inicial;
- $\delta : Q \times A \rightarrow Q$  é uma função que determina a transição entre estados; e
- $F \subseteq Q$  é um conjunto dos estados de aceitação.

Graficamente um AFD pode ser representado por um grafo dirigido, onde os nós correspondem aos estados e as setas às transições. Visualmente, é habitual usarem-se círculos para representar os estados. Os círculos correspondentes aos estados de aceitação são diferenciados usando-se riscos duplos. O estado inicial é marcado com uma pequena seta sem origem. As setas são etiquetadas com elementos do alfabeto  $A$ . De cada estado tem de sair uma e uma só seta etiquetada com cada elemento do alfabeto. Duas ou mais transições entre o mesmo par de estados são representadas por uma única seta, tendo como etiqueta uma lista de elementos do alfabeto.

### Exemplo 5.1

A figura seguinte



representa um autómato finito determinista  $M = (A, Q, q_0, \delta, F)$ , com

- $A = \{0, 1\}$ ,

- $Q = \{A, B\}$ ,
- $q_0 = A$ ,
- $\delta(A, 0) = A, \delta(A, 1) = B, \delta(B, 0) = B, \delta(B, 1) = A$  e
- $F = \{B\}$ .

*↗ muito comum este tipo de grafos! ↘*

Este AFD reconhece o conjunto das sequências binárias com um número ímpar de uns. Os estados  $A$  e  $B$  representam respectivamente *número par de uns* e *número ímpar de uns*. Inicialmente, o AFD encontra-se em  $A$  – a sequência vazia tem um número par de uns. A seguir, por cada 1 que entra, o AFD transita entre os estados  $A$  e  $B$ , de modo a refletir o número de uns lidos até esse momento. A chegada de zeros não altera o estado. Quando a palavra de entrada se esgotar, se o AFD se encontrar em  $A$  é porque tinha um número par de uns e é por isso rejeitada; se se encontrar em  $B$  é aceite.

Por exemplo:

- A palavra 1011 faz  $M_1$  evoluir de  $A$  para  $B$  ( $A \xrightarrow{1} B \xrightarrow{0} B \xrightarrow{1} A \xrightarrow{1} B$ ), aceitando-a como sendo uma palavra pertencente à linguagem reconhecida por  $M_1$ .
- A palavra 1100 faz  $M_1$  evoluir de  $A$  para  $A$  ( $A \xrightarrow{1} B \xrightarrow{1} A \xrightarrow{0} A \xrightarrow{0} A$ ), levando à sua rejeição.
- A palavra vazia,  $\varepsilon$ , deixa  $M_1$  em  $A$ , levando à sua rejeição.

O exemplo anterior mostra que um AFD também pode ser representado de forma textual. Sendo o alfabeto e o conjunto de estados conjuntos finitos, a função de transição é também um conjunto finito, que pode ser representada por um conjunto de triplos  $(s_1, a, s_2)$ , sendo  $s_2 = \delta(s_1, a)$ . A função de transição pode também ser representada por uma matriz, em que as linhas correspondem ao estado atual, as colunas aos símbolos do alfabeto e as células contêm o estado seguinte. Isto é ilustrado pelo exemplo seguinte.

*↗ aproximação com os gráficos regulares ... ↘*

### Exemplo 5.2

O AFD do exemplo anterior pode ser textualmente representado por

$$A = \{0, 1\}$$

$$Q = \{A, B\}$$

$$q_0 = A$$

$$F = \{B\}$$

$$\delta = \{(A, 0, A), (A, 1, B), (B, 0, B), (B, 1, A)\}$$

A função de transição  $\delta$  pode ser representada matricialmente por

$$\begin{array}{c|cc} & 0 & 1 \\ \hline A & A & B \\ B & B & A \end{array} \quad \delta: Q \times A \rightarrow Q$$

*↗ ↘*

## 5.2 Linguagem reconhecida por um AFD

Diz-se que um AFD  $M = (A, Q, q_0, \delta, F)$  **aceita** uma palavra  $u \in A^*$  se  $u$  se puder escrever na forma  $u = u_1 u_2 \cdots u_n$  e existir uma sequência de estados  $s_0, s_1, \dots, s_n$ , que satisfaça as seguintes condições:

1.  $s_0 = q_0$ ;
2. qualquer que seja o  $i = 1, \dots, n$ ,  $s_i = \delta(s_{i-1}, u_i)$ ;
3.  $s_n \in F$ .

Caso contrário diz-se que  $M$  **rejeita** a sequência de entrada.

### Exemplo 5.3

Usando o autômato do exemplo 5.1, mostre que a palavra 1101 é reconhecida por esse autômato.

**Resposta:**

$\delta(A, 1) = B, \delta(B, 1) = A, \delta(A, 0) = A$  e  $\delta(A, 1) = B$ , logo 1101 é reconhecida por  $M_1$ .

Seja  $\delta^* : Q \times A^* \rightarrow Q$  a extensão de  $\delta$  definida indutivamente por:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, av) = \delta^*(\delta(q, a), v), \quad \text{com } a \in A \wedge v \in A^*$$

$M$  aceita a palavra  $u \in A^*$  se e só se  $\delta^*(q_0, u) \in F$ .

A linguagem reconhecida por  $M = (A, Q, q_0, \delta, F)$  denota-se por  $L(M)$  e é definida por

$$L(M) = \{u \in A^* \mid M \text{ aceita } u\} = \{u \in A^* \mid \delta^*(q_0, u) \in F\}$$

### Exemplo 5.4

Usando  $\delta^*$  verifique se a palavra  $u = abab$  é aceita pelo autômato  $M = (A, Q, q_0, \delta, F)$ , definido por:

$$A = \{a, b, c\}$$

$$Q = \{s_1, s_2\}$$

$$q_0 = s_1$$

$$F = \{s_1\}$$

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

**Resposta:** Pretende-se verificar se  $\delta^*(s_1, abab) \in F$ .

$$\delta^*(s_1, abab) = \delta^*(\delta(s_1, a)bab) = \delta^*(s_2, bab)$$

$$= \delta^*(\delta(s_2, b)ab) = \delta^*(s_2, ab)$$

$$= \delta^*(\delta(s_2, a)b) = \delta^*(s_1, b)$$

$$= \delta^*(\delta(s_1, b)\varepsilon) = \delta^*(s_1, \varepsilon)$$

$$= s_1.$$



Como  $s_1 \in F$ ,  $M$  aceita a palavra  $abab$ .

### 5.3 Projeto de um AFD (Projetar um AFD que reconheça uma linguagem)

O projeto de um autômato finito é um processo criativo; como tal não pode ser reduzido a uma simples receita ou fórmula. Pode, no entanto, ajudar se se seguir os seguintes passos:

1. Identificar os estados necessários.
2. Acrescentar as transições.
3. Identificar e marcar o estado inicial.
4. Identificar e marcar os estados de aceitação.

Frequentemente, é preciso iterar diversas vezes sobre estes passos antes de se chegar à solução para o problema.

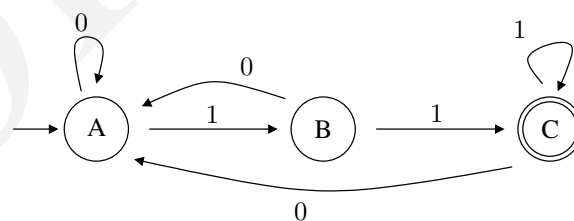
#### Exemplo 5.5

Projete um AFD que reconheça as sequências binárias terminadas em 11.

**Resposta:** Depois de alguma reflexão chega-se à conclusão de que bastam 3 estados,  $A$ ,  $B$  e  $C$ :

- $A$  significando que o último dígito que entrou não é 1;
- $B$  significando que o último dígito que entrou é 1 mas o anterior não o é;
- $C$  significando que os dois últimos dígitos entrados são 1.

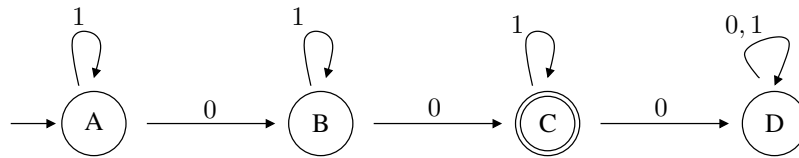
A partir daqui chega-se facilmente ao seguinte autômato



#### Exemplo 5.6

Projete um AFD que reconheça as sequências binárias com exatamente dois zeros e qualquer número de uns.

**Resposta:** Na figura seguinte apresenta-se graficamente o AFD pretendido.

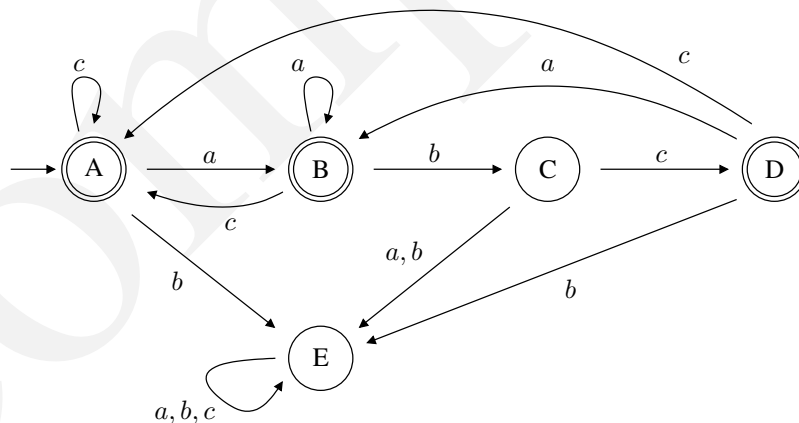


São necessários 4 estados para representar a chegada de 0, 1, 2 e mais de 2 zeros. Note que do estado  $D$  saem duas setas dirigidas ao próprio estado  $D$ , um com a etiqueta 0 e outro com a etiqueta 1. Por uma questão de simplificação notacional, é costume fundir-se as duas setas numa só, com as etiquetas separadas por vírgulas. Note ainda que num autômato finito determinista de cada estado deve sair uma seta etiquetada com cada um dos símbolos do alfabeto, mesmo que essas setas já não possam conduzir a situações de aceitação. É o caso das setas saídas do estado  $D$ .

### Exemplo 5.7

Projete um AFD que reconheça as sequências definidas sobre o alfabeto  $A = \{a, b, c\}$  que satisfazem o requisito de qualquer  $b$  ter um  $a$  imediatamente à sua esquerda e um  $c$  imediatamente à sua direita.

**Resposta:** À partida, podemos considerar a existência de 4 estados para detetar a sequência  $abc$ . Isto corresponde aos estados  $A, B, C$  e  $D$  da figura abaixo e as setas entre eles indo da esquerda para a direita. Basta que um  $b$  não satisfaça o requisito imposto para que a sequência seja rejeitada. O estado  $E$  captura essas situações.



Note que os estados  $A, B$  e  $D$  são de aceitação. O estado  $C$  não o é porque se o autômato termina aí, a sequência de entrada termina em  $b$  e, por conseguinte, existe um  $b$ , esse último, que não tem um  $c$  imediatamente à sua direita. Veremos adiante que os estados  $A$  e  $D$  são equivalentes podendo ser fundidos. No entanto, isso não invalida que o AFD dado esteja correto.

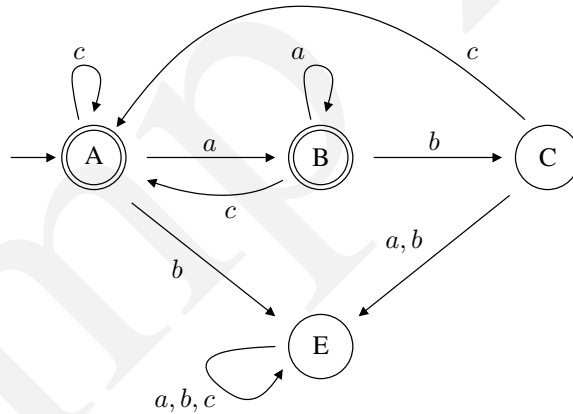
**Exercício 5.1**

Projete uma AFD que reconheça sequências binárias de comprimento ímpar terminadas em 0 ou de comprimento par terminadas em 1.

Dica: é possível definir-se um autômato com 5 estados.

**5.4 AFD reduzido**

Considere o autômato do exemplo anterior (exemplo 5.7). Os estados  $A$  e  $D$  são equivalentes. Por um lado, a sequência de entrada é aceite se o autômato termina nos estados  $A$  ou  $D$ . Por outro lado, se o autômato se encontra nos estados  $A$  ou  $D$  evolui, em ambos os casos, para o estado  $B$  se entra um  $a$ , para o  $E$  se entra um  $b$  e para o  $A$  se entra um  $c$ . Ou seja, enviar o autômato para o estado  $D$  é equivalente a enviá-lo para o estado  $A$ , podendo, por isso, um ser substituído pelo outro. Se desviarmos para  $A$  a única seta dirigida para  $D$  — vindo de  $C$  com etiqueta  $c$  —, o estado  $D$  deixa de ser alcançável a partir do estado inicial, podendo ser eliminado. O autômato transforma-se então no autômato da figura abaixo, que lhe é equivalente.



Em geral, dois estados,  $s_i$  e  $s_j$ , de um autômato  $M = (A, Q, q_0, \delta, F)$  são equivalentes se e só se

$$\forall u \in A^* \quad \delta^*(s_i, u) \in F \Leftrightarrow \delta^*(s_j, u) \in F$$

A notação  $s_i \equiv s_j$  é usada para representar o facto de  $s_i$  e  $s_j$  serem equivalentes. O conjunto de todos os estados equivalentes a  $s$  é denotado por  $[s]$  e representa uma classe de equivalência. A relação de equivalência ( $\equiv$ ) entre todos os estados de um autômato determinista permite definir o seu equivalente **reduzido**.

Seja  $M = (A, Q, q_0, \delta, F)$  um autômato determinista qualquer. O equivalente reduzido de  $M$  é o AFD  $M' = (A, Q', q'_0, \delta', F')$  definido da seguinte maneira:

- $Q' = \{[q] \mid q \in Q\}$

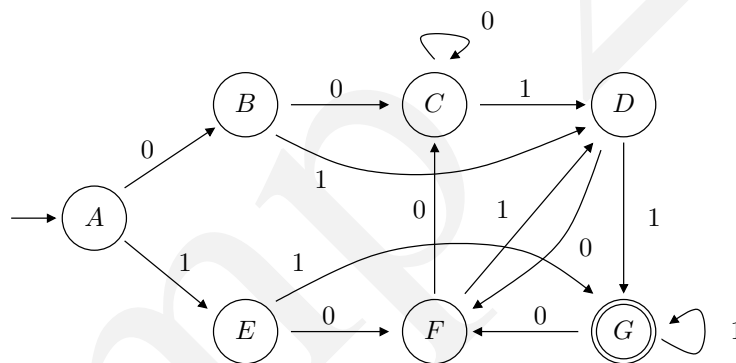
- $q'_0 = [q_0]$
- $F' = \{[q] \mid q \in F\}$
- $\delta' : Q' \times A \rightarrow Q'$  tal que  $\delta'([q], a) = [\delta(q, a)]$ .

### 5.4.1 Algoritmo de redução de AFD

O algoritmo de redução usa um método de aproximações sucessivas a partir de uma partição inicial dos estados em dois conjuntos – candidatos a classes de equivalência –, um com os estados de aceitação e outro com os restantes. Um dado conjunto  $C$  de estados é uma classe de equivalência se e só se para qualquer elemento  $a$  do alfabeto e qualquer par de estados  $q_1$  e  $q_2$  pertencentes a  $C$ ,  $[\delta(q_1, a)] = [\delta(q_2, a)]$ . Sempre que um conjunto viole a condição anterior, é desdobrado em dois ou mais conjuntos e repete-se o processo, até atingir uma solução, que existe sempre.

#### Exemplo 5.8

Considere o autômato



que aceita sequências binárias terminadas em 11, sendo, por isso, equivalente ao autômato apresentado no exemplo 5.5. Construa-se o seu equivalente reduzido.

**Resposta:** Primeiro, definem-se dois conjuntos de estados, um com os estados de aceitação e outro com os restantes

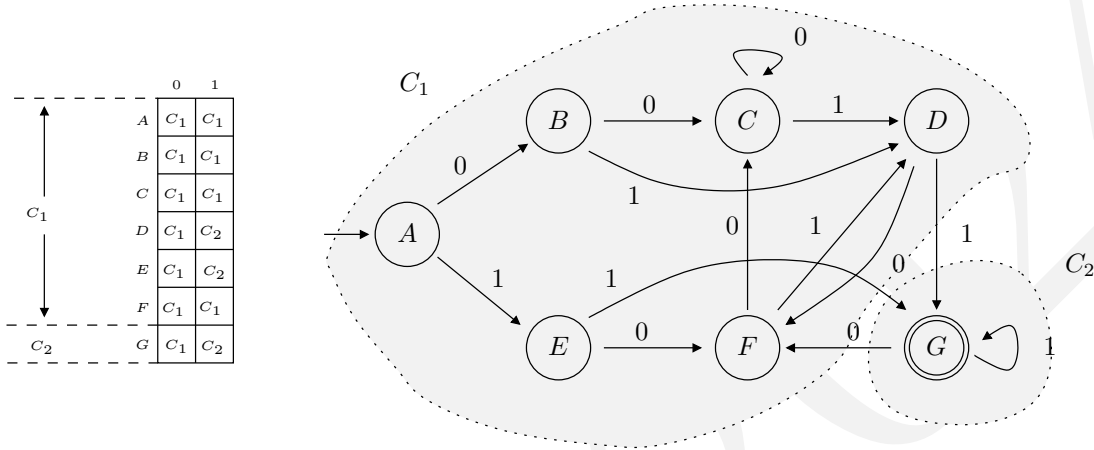
$$C_1 = Q - F = \{A, B, C, D, E, F\}$$

$$C_2 = F = \{G\}$$

Seja  $Q'$  o conjunto desses candidatos a classes de equivalência, i.e.,  $Q' = \{C_1, C_2\}$ .

A seguir, para cada elemento de  $Q'$ , avalia-se a sua evolução em termos dos elementos de  $Q'$  alcançados em consequência das possíveis transições. Isso está ilustrado na tabela à esquerda, na figura abaixo. Como se pode constatar, os estados de  $C_1$  transitam para estados de  $C_1$ , por ocorrência de um 0. Mas, por ocorrência de um 1, há estados que transitam para estados de  $C_1$  e outros que transitam para estados de  $C_2$ . Isto significa que o conjunto de estados  $C_1$  não é uma classe de equivalência. A mesma constatação pode ser feita através da representação em grafo,

ilustrado à direita na mesma figura. Nos estados de  $C_1$ , há transições em 1 que se mantêm em  $C_1$  e outras que se dirigem a  $C_2$ .  $C_2$ , porque tem apenas um elemento, é obviamente uma classe de equivalência.



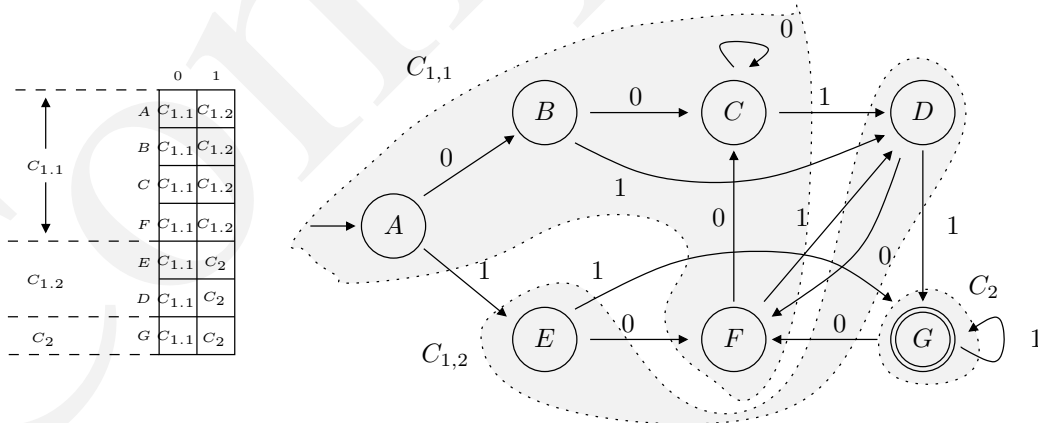
O conjunto  $C_1$ , não sendo uma classe de equivalência, tem de ser desdobrado. Olhando para a tabela, verifica-se que as linhas referentes a  $C_1$  caem em duas categorias, uma com os estados A, B, C e F e outra com os estados D e E. Divide-se então o conjunto  $C_1$  em dois subconjuntos, representando estas duas categorias, passando  $Q'$  a ter 3 elementos:

$$C_{1.1} = \{A, B, C, F\}$$

$$C_{1.2} = \{D, E\}$$

$$C_2 = \{G\}$$

Reconstruindo a tabela e o grafo, obtém-se



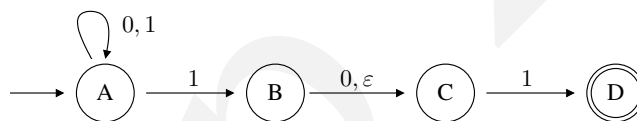
Pode facilmente constatar-se que  $C_{1.1}$  e  $C_{1.2}$  são classes de equivalência, pelo que o processo de redução terminou. Se tal não acontecesse, os conjuntos de estados que não fossem classes de equivalência deveriam ser desdobrados, repetindo-se o processo enquanto necessário. Pode verificar que o autómato obtido é isomorfo com o apresentado no exemplo 5.5.

O algoritmo encontra sempre uma solução. Se se tentar reduzir um autômato finito determinista já reduzido, o processo de redução conduzirá ao próprio autômato (na realidade, a um que lhe é isomorfo).

## Capítulo 6

# Autómatos Finitos Não Deterministas (AFND)

Considere o seguinte autômato definido sobre o alfabeto  $A = \{0, 1\}$ .



Possui 3 características diferentes das assumidas na definição de autômato finito determinista:

- Não há nenhuma seta etiquetada com 1 a sair dos estados  $B$  e  $D$ , nem nenhuma seta etiquetada com 0 a sair dos estados  $C$  e  $D$ .
- Há duas setas etiquetadas com 1 a sair do estado  $A$ , um para  $A$  e outro para  $B$ .
- Há uma seta etiquetada com  $\varepsilon$ , a palavra vazia, a sair do estado  $B$ . Designar-se-ão este tipo de transições por **transições- $\varepsilon$** .

Estas características violam a definição de autômato finito determinista, mas são aceites pelos autómatos finitos não deterministas (AFND), que permitem que de cada estado possam sair zero ou mais setas etiquetadas com cada um dos símbolos do alfabeto de entrada ou com a palavra vazia. A multiplicidade permite que para fazer aceitar uma palavra se possa escolher entre caminhos alternativos. Para que uma palavra seja aceite basta que exista um caminho que conduza a um estado de aceitação, mesmo que haja outros que conduzam a estados de não aceitação. Perante a entrada 1011 o autômato anterior pode realizar 4 caminhos alternativos:

$$\begin{aligned}
A &\xrightarrow{1} A \xrightarrow{0} A \xrightarrow{1} A \xrightarrow{1} A \\
A &\xrightarrow{1} A \xrightarrow{0} A \xrightarrow{1} A \xrightarrow{1} B \\
A &\xrightarrow{1} A \xrightarrow{0} A \xrightarrow{1} A \xrightarrow{1} B \xrightarrow{\varepsilon} C \\
A &\xrightarrow{1} A \xrightarrow{0} A \xrightarrow{1} B \xrightarrow{\varepsilon} C \xrightarrow{1} D
\end{aligned}$$

um dos quais, o último, conduz ao estado de aceitação. Logo, porque há pelo menos um caminho que termina num estado de aceitação, a palavra 1011 é aceite. Este autómato reconhece todas as sequências binárias terminadas em 11 ou 101.

## 6.1 Definição de autómato finito não determinista

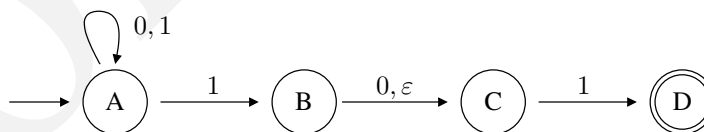
Um **autómato finito não determinista** (AFND) é um quintuplo  $M = (A, Q, q_0, \delta, F)$ , em que:

- $A$  é o alfabeto de entrada;
- $Q$  é um conjunto finito não vazio de estados;
- $q_0 \in Q$  é o estado inicial;
- $\delta \subseteq (Q \times A_\varepsilon \times Q)$  é a relação de transição entre estados, com  $A_\varepsilon = A \cup \{\varepsilon\}$ ; e
- $F \subseteq Q$  é o conjunto dos estados de aceitação.

A diferença relativamente à definição de autómato finito determinista verifica-se na relação  $\delta$ . Por um lado, permite que as setas (transições) sejam etiquetadas com  $\varepsilon$ . Por outro lado, pelo facto de  $\delta$  ser uma relação e não uma função, permite que o mesmo par  $(q, a) \in Q \times A_\varepsilon$  possa ter 0 ou mais imagens.

### Exemplo 6.1

À luz da definição anterior, apresente os vários elementos do AFND graficamente representado na figura seguinte.



#### Resposta:

Considerando que o AFND é representado pelo tuplo  $M = (A, Q, q_0, \delta, F)$ , tem-se

$$A = \{0, 1\}$$

$$Q = \{A, B, C, D\}$$

$$q_0 = A$$

$$F = \{D\}$$

$$\delta = \{(A, 0, A), (A, 1, A), (A, 1, B), (B, \varepsilon, C), (B, 0, C), (C, 1, D)\}$$



Note que a existência dos elementos  $(A, 1, A)$  e  $(A, 1, B)$  faz com que  $\delta$  não seja uma função.

Alternativamente, é possível definir-se a relação de transição entre estados como uma função  $\Delta$  que aplica o conjunto  $Q \times A_\epsilon$  em  $\wp(Q)$ , onde  $\wp(Q)$  representa o conjunto dos subconjuntos de  $Q$ .

### Exemplo 6.2

Usando esta última definição para a relação de transição entre estados, apresente os vários elementos do AFND graficamente representado no exemplo anterior (exemplo 6.1).

#### Resposta:

Considerando que o AFND é representado pelo tuplo  $M = (A, Q, q_0, \delta, F)$ , tem-se

$$A = \{0, 1\}$$

$$Q = \{A, B, C, D\}$$

$$q_0 = A$$

$$\Delta = \left\{ \begin{array}{lll} (A, 0) \rightarrow \{A\}, & (A, 1) \rightarrow \{A, B\}, & (A, \epsilon) \rightarrow \emptyset, \\ (B, 0) \rightarrow \{C\}, & (B, 1) \rightarrow \emptyset, & (B, \epsilon) \rightarrow \{C\}, \\ (C, 0) \rightarrow \emptyset, & (C, 1) \rightarrow \{D\}, & (C, \epsilon) \rightarrow \emptyset, \\ (D, 0) \rightarrow \emptyset, & (D, 1) \rightarrow \emptyset, & (D, \epsilon) \rightarrow \emptyset \end{array} \right\}$$

$$F = \{D\}$$

11

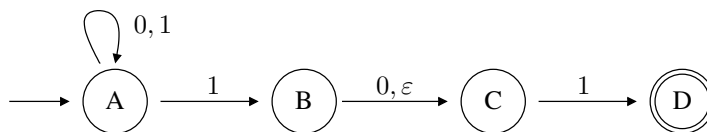
Neste caso,  $\Delta$  é uma função que aplica  $Q \times A_\epsilon \rightarrow \wp(Q)$  e, como  $Q$  tem 4 elementos e  $A_\epsilon = \{0, 1, \epsilon\}$  tem 3,  $\Delta$  tem 12 elementos. Note que sendo  $\Delta$  definida como uma função é preciso apresentar as imagens para todos os elementos de  $Q \times A_\epsilon$ , mesmo que sejam o conjunto vazio. Em geral, torna-se mais clara a representação usando a relação de transição.

## 6.2 Árvore de caminhos

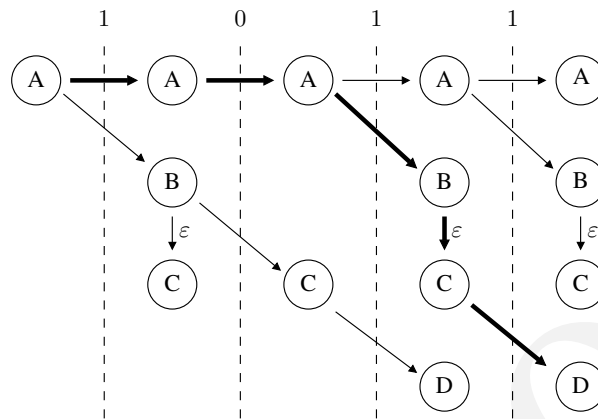
Uma forma simples de obter os diversos caminhos que um AFND pode realizar em resposta a uma dada palavra à entrada é traçando a *árvore de caminhos*. Corresponde a um grafo acíclico, em forma de árvore, tendo como raiz o estado inicial e onde se representam todos os estados alcançáveis por ocorrência da palavra de entrada.

### Exemplo 6.3

Determine a árvore de caminhos que representa a resposta do autómato abaixo à palavra 1011.



**Resposta:**



A árvore de caminhos é uma estrutura em camadas (subconjunto de estados) alcançadas por força da ocorrência de cada símbolo da palavra de entrada. Uma palavra é aceita se o subconjunto de estados da última camada contiver pelo menos um estado de aceitação, ou seja, se a intersecção com o conjunto de aceitação não for o conjunto vazio. No exemplo anterior, o caminho que permite aceitar a palavra foi posto em realce. No exemplo, tenha também em atenção a forma como as transições- $\varepsilon$  são tratadas. Elas ficam dentro da mesma camada (subconjunto) uma vez que não correspondem à ocorrência de um símbolo do alfabeto.

## 6.3 Linguagem reconhecida por um AFND

Diz-se que um AFND  $M = (A, Q, q_0, \delta, F)$ , **aceita** uma palavra  $u \in A^*$  se e só se  $u$  se puder escrever na forma  $u = u_1 u_2 \cdots u_n$ , com  $u_i \in A_\varepsilon$ , e existir uma sequência de estados  $s_0, s_1, \dots, s_n$ , que satisfaça as seguintes condições:

1.  $s_0 = q_0$ ;
2. qualquer que seja o  $i = 1, \dots, n$ ,  $(s_{i-1}, u_i, s_i) \in \delta$ ;
3.  $s_n \in F$ .

Caso contrário diz-se que  $M$  **rejeita** a entrada. Note que, nos autómatos finitos não deterministas,  $n$  pode ser maior que  $|u|$ , porque alguns dos  $u_i$  podem ser  $\varepsilon$ . Isto está patente no exemplo anterior (exemplo 6.3) no caminho de reconhecimento da palavra 1011.

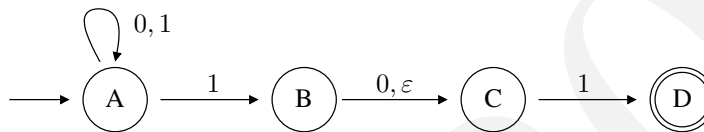
A linguagem reconhecida por  $M$  denota-se por  $L(M) = \{u \mid M \text{ aceita } u\}$ . Usar-se-á a notação  $q_i \xrightarrow{u} q_j$  para representar a existência de uma palavra  $u$  que conduza do estado  $q_i$  ao estado  $q_j$ . Usando esta notação tem-se  $L(M) = \{u \mid q_0 \xrightarrow{u} q_f \wedge q_f \in F\}$ .

## 6.4 Fecho- $\varepsilon$

O conceito de fecho- $\varepsilon$  ( $\varepsilon$ -closure, em inglês) é central à manipulação de AFND. Dado um AFND  $M = (A, Q, q_0, \delta, F)$  qualquer,  $\varepsilon\text{-closure} : Q \rightarrow \wp(Q)$  é uma função que associa a um dado estado  $q \in Q$  o subconjunto de estados direta ou indiretamente alcançáveis a partir de  $q$  apenas por transições- $\varepsilon$ , incluindo o próprio estado  $q$ .

### Exemplo 6.4

Determine o  $\varepsilon\text{-closure}()$  de cada um dos estados do AFND seguinte.



**Resposta:**

$$\varepsilon\text{-closure}(A) = \{A\}$$

$$\varepsilon\text{-closure}(B) = \{B, C\}$$

$$\varepsilon\text{-closure}(C) = \{C\}$$

$$\varepsilon\text{-closure}(D) = \{D\}$$

$$\varepsilon\text{-closure}(A, B) = \varepsilon\text{-closure}(A) \cup \varepsilon\text{-closure}(B) = \{A, B, C\}$$

É conveniente estender-se a definição de fecho- $\varepsilon$ , considerando que o argumento da função é um subconjunto de estados em vez de apenas um. Neste caso, o resultado é a união dos fechos de cada um dos estados.

## Capítulo 7

# Equivalência entre AFD e AFND

A definição de AFND incorpora a definição de AFD, pelo que qualquer AFD é por definição um AFND. Na verdade, na definição de AFD,  $\delta$  é uma função que aplica  $Q \times A$  em  $Q$ , logo  $\delta \subset (Q \times A \times Q)$ . Mas, sendo  $A \subset A_\epsilon$ ,  $(Q \times A \times Q) \subset (Q \times A_\epsilon \times Q)$ , pelo que a função  $\delta$  dos AFD é um subconjunto da relação  $\delta$  dos AFND.

Trivial  
↓ O problema é ao contrário...

### 7.1 Conversão de AFND em AFD

Prova-se também que dado um qualquer AFND é possível construir-se um AFD que reconhece exatamente a mesma linguagem. Olhando para uma árvore de caminhos, constata-se que por ação de um símbolo à entrada o autômato não determinista avança de um subconjunto de estados para outro, sendo que inicialmente se encontra no subconjunto constituído pelo estado inicial mais aqueles direta ou indiretamente alcançáveis por transições- $\epsilon$ , ou seja, o fecho- $\epsilon$  do estado inicial. É assim possível transformar um AFND num AFD cujos estados são subconjuntos de estados do AFND. Se o AFND tiver  $n$  estados, haverá  $2^n$  subconjuntos de estados, pelo que o AFD equivalente terá no máximo  $2^n$  estados. Ver-se-á adiante que, em geral, muitos destes estados não são alcançáveis a partir do estado inicial, pelo que podem ser descartados. Os estados de aceitação do AFD serão todos aqueles que contenham estados de aceitação do AFND, ou seja, aqueles cuja intersecção com o conjunto de aceitação do AFND não seja o conjunto vazio.

A estrutura de construção da árvore de caminhos permite obter a resposta de um AFND a cada símbolo do alfabeto de entrada, considerando que o AFND se encontra hipoteticamente num dado subconjunto de estados. É assim possível, dado um AFND  $M = (A, Q, q_0, \delta, F)$ , definir-se uma função  $\delta' : \wp(Q) \times A \rightarrow \wp(Q)$ , que representa as transições do autômato em termos de subconjuntos de estados.

Dado um AFND  $M = (A, Q, q_0, \delta, F)$  qualquer, o autômato  $M' = (A, Q', q'_0, \delta', F')$  onde:

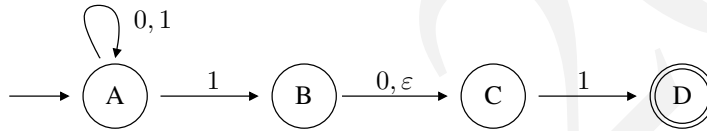
- $Q' = \wp(Q)$ ;

- $\delta' : \wp(Q) \times A \rightarrow \wp(Q)$ , com  $\delta'(p', a) = \bigcup_{p \in p'} (\varepsilon\text{-closure}(\Delta(p, a)))$
- $q'_0 = \varepsilon\text{-closure}(q_0)$ ; e
- $F' = \{q' : q' \in \wp(Q) \wedge q' \cap F \neq \emptyset\}$

é um autômato finito determinista equivalente a  $M$ . Nesta definição, optou-se por usar a função  $\Delta$  e não a relação  $\delta$ , porque simplifica a formulação da função  $\delta'$ . Note que, na definição de  $\delta'$ ,  $p'$  representa um elemento de  $\wp(Q)$ , sendo portanto um subconjunto de estados de  $Q$ . Relembro que um subconjunto de estados é de aceitação no autômato determinista equivalente desde que contenha pelo menos um estado de aceitação do autômato inicial.

### Exemplo 7.1

Usando o método anterior, construa-se um AFD equivalente ao AFND apresentado no exemplo 6.1, cuja representação gráfica se repete aqui por comodidade.



**Resposta:** Seja  $M' = (A, Q', q'_0, \delta', F')$  o autômato pretendido. Tem-se

$$Q' = \{X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}\}$$

onde

$$\begin{array}{llll} X_0 = \{\} & X_1 = \{A\} & X_2 = \{B\} & X_3 = \{A, B\} \\ X_4 = \{C\} & X_5 = \{A, C\} & X_6 = \{B, C\} & X_7 = \{A, B, C\} \\ X_8 = \{D\} & X_9 = \{A, D\} & X_{10} = \{B, D\} & X_{11} = \{A, B, D\} \\ X_{12} = \{C, D\} & X_{13} = \{A, C, D\} & X_{14} = \{B, C, D\} & X_{15} = \{A, B, C, D\} \end{array}$$

Tem-se ainda  $q'_0 = \varepsilon\text{-closure}(A) = X_1$  e

$$F' = \{X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}\}$$

Finalmente, a função  $\delta'$  é definida pela tabela seguinte

estado	0	1	estado	0	1	estado	0	1	estado	0	1
$X_0$	$X_0$	$X_0$	$X_1$	$X_1$	$X_7$	$X_2$	$X_4$	$X_0$	$X_3$	$X_5$	$X_7$
$X_4$	$X_0$	$X_8$	$X_5$	$X_1$	$X_{15}$	$X_6$	$X_4$	$X_8$	$X_7$	$X_5$	$X_{15}$
$X_8$	$X_0$	$X_0$	$X_9$	$X_1$	$X_7$	$X_{10}$	$X_4$	$X_0$	$X_{11}$	$X_5$	$X_7$
$X_{12}$	$X_0$	$X_8$	$X_{13}$	$X_1$	$X_{15}$	$X_{14}$	$X_4$	$X_8$	$X_{15}$	$X_5$	$X_{15}$

No exemplo anterior, observando a tabela de transição, é fácil constatar-se que a partir do estado inicial,  $X_1$ , apenas os estados  $X_7$ ,  $X_5$  e  $X_{15}$  são alcançáveis, pelo que os restantes podem ser retirados do

autômato. A obtenção de um AFD equivalente sem estados não alcançáveis (desnecessários) pode fazer-se por um método construtivo. A ideia é ir contruindo a função de transição  $\delta'$  e o conjunto de estados  $Q'$  a partir do estado inicial. Sendo  $q_0$  o estado inicial do AFND, o estado inicial do AFD equivalente é  $q'_0 = \varepsilon\text{-closure}(q_0)$  e o seu conjunto de estados começa em  $Q' = \{q'_0\}$ . A partir deste podem calcular-se os estados alcançados por ocorrência dos vários símbolos do alfabeto, sendo acrescentados a  $Q'$ . O processo repete-se até que todos os elementos de  $Q'$  estejam processados e nenhum novo surja.

O algoritmo seguinte consagra este procedimento. Nele, considera-se a definição de AFND baseada na função  $\Delta$ , função que, dados um estado e uma letra do alfabeto, devolve o conjunto de estados alcançados ( $\Delta : Q \times A \rightarrow \wp(Q)$ ), e a função  $\varepsilon\text{-closure}(\cdot)$  que recebe como entrada um subconjunto de estados.

### Algoritmo 7.1

NFA-to-DFA (input  $(A, Q, q_0, \delta, F)$ , output  $(A, Q', q'_0, \delta', F')$ ):

```

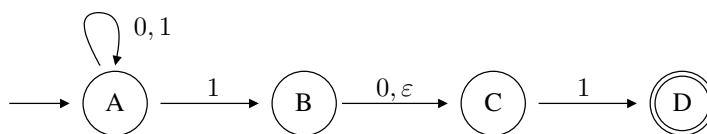
 $q'_0 \leftarrow \varepsilon\text{-closure}(\{q_0\})$ 
 $Q' \leftarrow \{q'_0\}$ 
 $O' \leftarrow Q'$ 
while  $O' \neq \emptyset$  do :
     $o' \leftarrow \text{take one element out of } O'$ 
    foreach  $a \in A$  do :
         $q' = \emptyset$ 
        foreach  $o \in o'$ 
             $q' \leftarrow q' \cup \varepsilon\text{-closure}(\Delta(o, a))$ 
         $\delta'(q', a) \leftarrow q'$ 
        if  $q' \notin Q'$  then :
             $Q' \leftarrow Q' \cup q'$ 
             $O' \leftarrow O' \cup q'$ 
 $F' \leftarrow \emptyset$ 
foreach  $q' \in Q'$  do :
    if  $(q' \cap F) \neq \emptyset$  then :
         $F' \leftarrow F' \cup q'$ 

```

O exemplo seguinte ilustra a aplicação deste algoritmo.

### Exemplo 7.2

Usando o método construtivo, construa-se um AFD equivalente ao AFND apresentado no exemplo 6.1, cuja representação gráfica se repete aqui por comodidade.



**Resposta:** Seja  $M' = (A, Q', q'_0, \delta', F')$  o autômato pretendido. Tem-se que

$$q'_0 = \varepsilon\text{-closure}(A) = \{A\} = X_1$$

De  $X_1$  saem duas transições, uma etiquetada com 0 e outra com 1.

$$\delta'(X_1, 0) = \varepsilon\text{-closure}(\Delta(A, 0)) = \varepsilon\text{-closure}(\{A\}) = \{A\} = X_1$$

$$\delta'(X_1, 1) = \varepsilon\text{-closure}(\Delta(A, 1)) = \varepsilon\text{-closure}(\{A, B\}) = \{A, B, C\} = X_7$$

Partindo de  $X_7$  tem-se

$$\begin{aligned} \delta'(X_7, 0) &= \varepsilon\text{-closure}(\Delta(A, 0)) \cup \varepsilon\text{-closure}(\Delta(B, 0)) \cup \varepsilon\text{-closure}(\Delta(C, 0)) \\ &= \varepsilon\text{-closure}(\{A\}) \cup \varepsilon\text{-closure}(\{C\}) \cup \varepsilon\text{-closure}(\{\}) = \{A\} \cup \{C\} \cup \{\} = X_5 \end{aligned}$$

$$\begin{aligned} \delta'(X_7, 1) &= \varepsilon\text{-closure}(\Delta(A, 1)) \cup \varepsilon\text{-closure}(\Delta(B, 1)) \cup \varepsilon\text{-closure}(\Delta(C, 1)) \\ &= \varepsilon\text{-closure}(\{A, B\}) \cup \varepsilon\text{-closure}(\{\}) \cup \varepsilon\text{-closure}(\{D\}) \\ &= \{A, B, C\} \cup \{\} \cup \{D\} = X_{15} \end{aligned}$$

Procedendo de forma equivalente para  $X_5$  e  $X_{15}$ , obtém-se

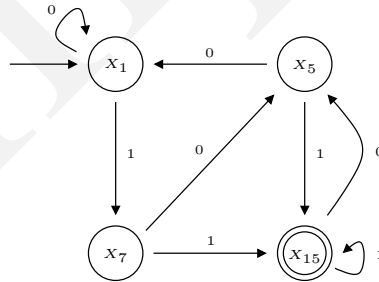
$$\delta'(X_5, 0) = \bigcup_{q \in X_5} (\varepsilon\text{-closure}(\Delta(q, 0))) = X_1$$

$$\delta'(X_5, 1) = \bigcup_{q \in X_5} (\varepsilon\text{-closure}(\Delta(q, 1))) = X_{15}$$

$$\delta'(X_{15}, 0) = \bigcup_{q \in X_{15}} (\varepsilon\text{-closure}(\Delta(q, 0))) = X_5$$

$$\delta'(X_{15}, 1) = \bigcup_{q \in X_{15}} (\varepsilon\text{-closure}(\Delta(q, 1))) = X_{15}$$

Sendo  $X_{15}$  o único estado que contém estados de aceitação do AFND, o AFD equivalente é o representado graficamente a seguir



## Capítulo 8

# Operações sobre AFD e AFND

A classe das linguagens regulares é fechada sob as operações de **reunião, concatenação, fecho de Kleene, complementação, diferença e intersecção**. Quer isto dizer que se  $M_1$  e  $M_2$  são dois autómatos finitos quaisquer, reconhecendo respetivamente as linguagens  $L(M_1)$  e  $L(M_2)$ , existem autómatos finitos que reconhecem as linguagens  $L(M_1) \cup L(M_2)$ ,  $L(M_1)L(M_2)$ ,  $L(M_1)^*$ ,  $\overline{L(M_1)}$ ,  $\underbrace{L(M_1) - L(M_2)}_{L(M_1) \cap L(M_2)}$  e  $L(M_1) \cap L(M_2)$ .

### 8.1 Reunião de autómatos finitos D e ND

Seja  $M_1 = (A, Q_1, q_1, \delta_1, F_1)$  e  $M_2 = (A, Q_2, q_2, \delta_2, F_2)$  dois autómatos (AFD ou AFND) quaisquer, e sejam  $L(M_1)$  e  $L(M_2)$  as linguagens por eles reconhecidas, respetivamente. O AFND  $M = (A, Q, q_0, \delta, F)$ , onde

$$Q = Q_1 \cup Q_2 \cup \{q_0\}, \quad \text{com } q_0 \notin Q_1 \wedge q_0 \notin Q_2$$

$$F = F_1 \cup F_2$$

$$\delta = \delta_1 \cup \delta_2 \cup \{(q_0, \varepsilon, q_1), (q_0, \varepsilon, q_2)\}$$

reconhece a linguagem  $L(M) = L(M_1) \cup L(M_2)$ .

Na figura 8.1 ilustra-se graficamente a construção de  $M$  a partir de  $M_1$  e  $M_2$ . Por um lado, é intuitivo que qualquer sequência  $u$  aplicada a  $M$ , segue não deterministicamente os caminhos de  $M_1$  e de  $M_2$ . Logo, se  $u$  é aceite por um deles também o é por  $M$ . Por outro lado, também é intuitivo que  $M$  só aceita sequências que sejam aceites por  $M_1$  ou por  $M_2$ . Mas, prove-se formalmente esta equivalência.

Pretende-se provar que  $L(M) = L(M_1) \cup L(M_2)$ . Para isso basta provar que

$$L(M) \subseteq L(M_1) \cup L(M_2)$$

e que

$$L(M_1) \cup L(M_2) \subseteq L(M)$$



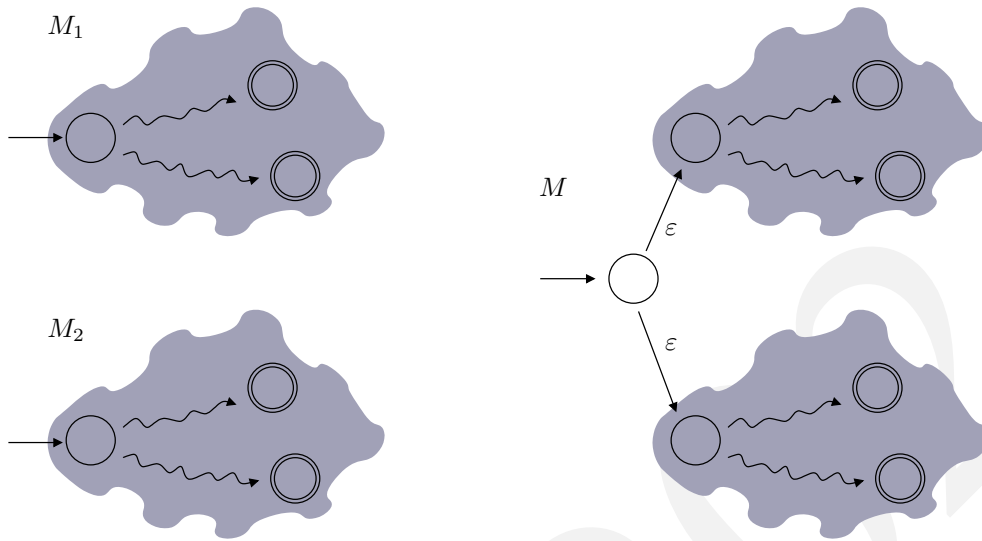


Figura 8.1: Ilustração gráfica da reunião de autômatos finitos.

A primeira condição corresponde a provar que

$$u \in L(M) \Rightarrow (u \in L(M_1) \vee u \in L(M_2))$$

o que se obtém do seguinte desenvolvimento

$$\begin{aligned} u \in L(M) &\Rightarrow q_0 \xrightarrow{u} q_f, \quad \text{com } q_f \in F \\ &\Rightarrow q_0 \xrightarrow{u} q_f, \quad \text{com } q_f \in F_1 \cup F_2 \\ &\Rightarrow (q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{u} q_f, \quad \text{com } q_f \in F_1) \vee (q_0 \xrightarrow{\varepsilon} q_2 \xrightarrow{u} q_f, \quad \text{com } q_f \in F_2) \\ &\Rightarrow (q_1 \xrightarrow{u} q_f, \quad \text{com } q_f \in F_1) \vee (q_2 \xrightarrow{u} q_f, \quad \text{com } q_f \in F_2) \\ &\Rightarrow u \in L(M_1) \vee u \in L(M_2) \end{aligned}$$

A segunda condição corresponde a provar, para  $i = 1, 2$ , que

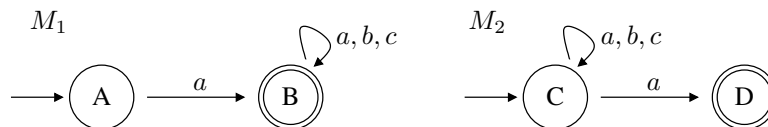
$$u \in L(M_i) \Rightarrow u \in L(M)$$

o que se obtém do seguinte desenvolvimento

$$\begin{aligned} u \in L(M_i) &\Rightarrow q_i \xrightarrow{u} q_f, \quad \text{com } q_f \in F_i \\ &\Rightarrow q_0 \xrightarrow{\varepsilon} q_i \xrightarrow{u} q_f, \quad \text{com } q_f \in F_i \\ &\Rightarrow q_0 \xrightarrow{u} q_f, \quad \text{com } q_f \in F_i \\ &\Rightarrow q_0 \xrightarrow{u} q_f, \quad \text{com } q_f \in F \\ &\Rightarrow u \in L(M) \end{aligned}$$

**Exemplo 8.1**

A figura



representa graficamente os autômatos  $M_1$  e  $M_2$ , definidos sobre o alfabeto  $A = \{a, b, c\}$ , que reconhecem respectivamente as linguagens

$$L_1 = \{aw | w \in A^*\} = \{w \in A^* | w \text{ começa por } a\}$$

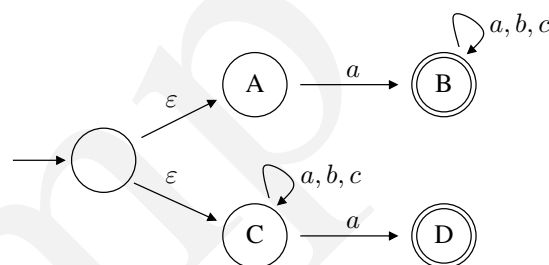
e

$$L_2 = \{wa | w \in A^*\} = \{w \in A^* | w \text{ termina por } a\}$$

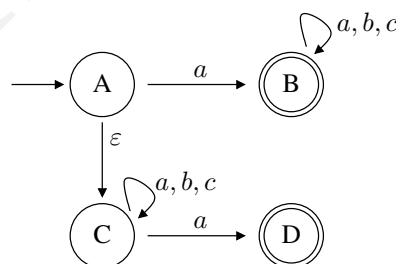
Usando a operação de reunião sobre autômatos determine o autômato  $M$  que reconhece a linguagem das palavras começadas ou terminadas por  $a$ .

**Resposta:**

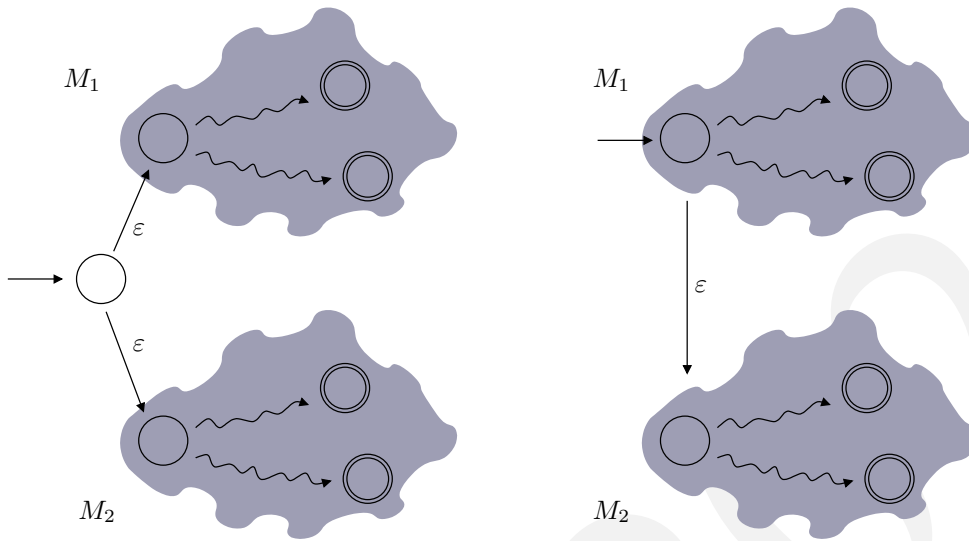
A aplicação direta do algoritmo de reunião resulta no autômato da figura abaixo



Alguma manipulação permite transformá-lo em

**Exercício 8.1**

A manipulação feita na resposta do exemplo anterior corresponde à ilustrada pelas duas composições da figura abaixo, definidas à volta dos autômatos  $M_1$  e  $M_2$  (a sombreado). Mostre que só são equivalente se o estado inicial de  $M_1$  não tiver setas a si dirigidas.



Considere que  $M_1$  e  $M_2$  são autômatos definidos sobre o alfabeto binário, sendo que  $M_1$  reconhece as sequências com número par de uns e  $M_2$  as sequências com número ímpar de zeros. Correspondendo à aplicação direta da reunião de autômatos finitos, a construção da esquerda reconhece as sequências com número par de uns ou ímpar de zeros. Por conseguinte, rejeita as sequências com número ímpar de uns e par de zeros. No entanto, estas sequências são aceites pelo autômato da direita. Mostre-o. Qual é a linguagem reconhecida pela construção da direita?

## 8.2 Concatenação de autômatos finitos

Seja  $M_1 = (A, Q_1, q_1, \delta_1, F_1)$  e  $M_2 = (A, Q_2, q_2, \delta_2, F_2)$  dois autômatos (AFD ou AFND) quaisquer, e sejam  $L(M_1)$  e  $L(M_2)$  as linguagens por eles reconhecidas, respetivamente. O AFND  $M = (A, Q, q_0, \delta, F)$ , onde

$$Q = Q_1 \cup Q_2$$

$$q_0 = q_1$$

$$F = F_2$$

$$\delta = \delta_1 \cup \delta_2 \cup (F_1 \times \{\varepsilon\} \times \{q_2\})$$

reconhece a linguagem  $L(M) = L(M_1).L(M_2)$ , concatenação de  $L_1$  com  $L_2$ .

Na figura 8.2 ilustra-se graficamente a construção de  $M$  a partir de  $M_1$  e  $M_2$ . Para que  $M$  aceite uma palavra  $u$  é necessário percorrer um caminho que vá desde o seu estado inicial até um dos seus estados de aceitação. Ora, isto significa ir do estado inicial até um estado de aceitação de  $M_1$ , passando-se, de seguida e por efeito de um  $\varepsilon$ , para o estado inicial de  $M_2$  e finalmente indo deste até um dos estados de aceitação de  $M_2$ . Ou seja, uma palavra  $u$  é aceite por  $M$  se se puder decompor  $u$  em duas partes  $v$  e  $w$  ( $u = vw$ ), sendo  $v$  aceite por  $M_1$  e  $w$  por  $M_2$ . Prove-se formalmente esta equivalência.

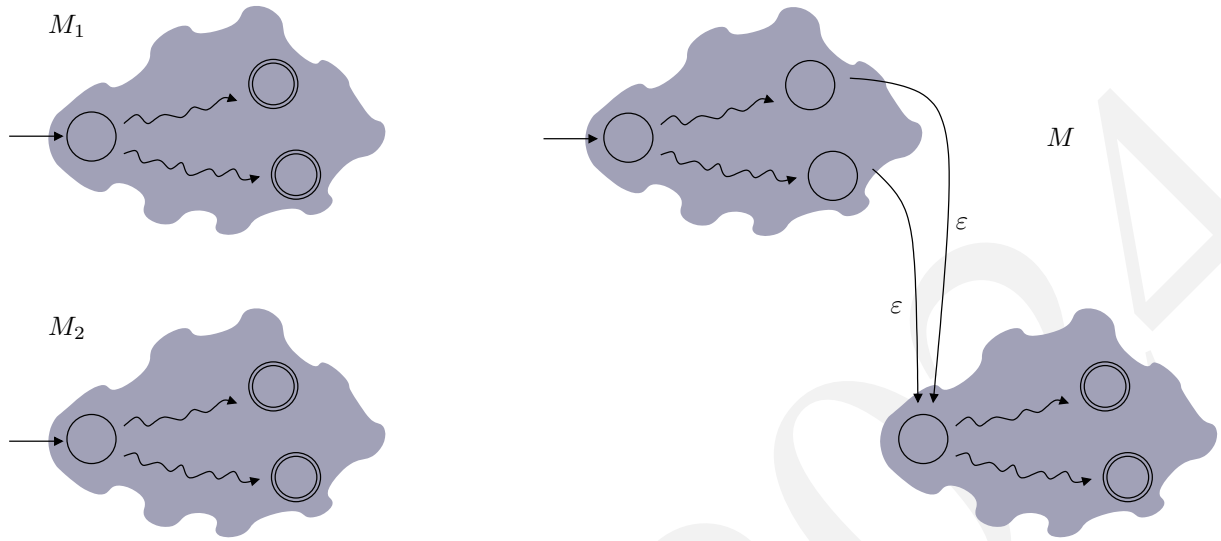


Figura 8.2: Ilustração gráfica da concatenação de autômatos finitos.

Pretende-se provar que  $L(M) = L(M_1).L(M_2)$ . Para isso basta provar que

$$L(M) \subseteq L(M_1).L(M_2)$$

e que

$$L(M_1).L(M_2) \subseteq L(M)$$

A primeira condição corresponde a provar que

$$u \in L(M) \Rightarrow (u \in L(M_1)L(M_2))$$

o que se obtém do seguinte desenvolvimento

$$u \in L(M) \Rightarrow q_0 \xrightarrow{u} q_f, \quad \text{com } q_f \in F$$

$$u \in L(M) \Rightarrow q_0 \xrightarrow{u} q_f, \quad \text{com } q_f \in F_2$$

$$\Rightarrow q_0 \xrightarrow{v} q_v \xrightarrow{\epsilon} q_2 \xrightarrow{w} q_f, \quad \text{com } u = vw \wedge q_v \in F_1 \wedge q_f \in F_2$$

$$\Rightarrow v \in L(M_1) \wedge w \in L(M_2), \quad \text{com } u = vw$$

$$\Rightarrow u \in L(M_1)L(M_2)$$

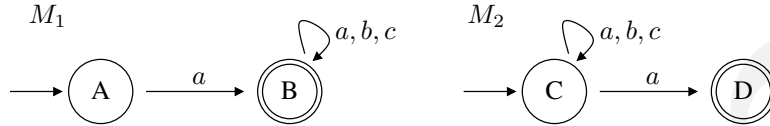
### Exercício 8.2

Prove a segunda condição, isto é, prove que

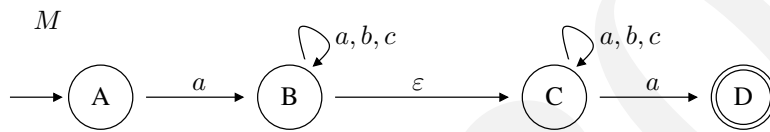
$$u \in L(M_1)L(M_2) \Rightarrow u \in L(M)$$

**Exemplo 8.2**

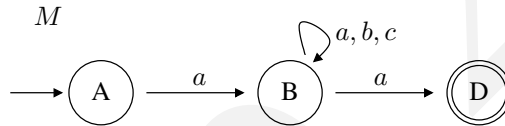
Usando o algoritmo apresentado, determine o autômato  $M$  que reconhece a linguagem  $L(M_1).L(M_2)$ , considerando os autômatos  $M_1$  e  $M_2$  do exemplo 8.1, que se repetem aqui por comodidade.



**Resposta:** A aplicação direta do algoritmo de concatenação resulta no autômato da figura abaixo



Note que  $B$  deixou de ser estado de aceitação. Alguma manipulação, permite que se chegue ao autômato

**Exercício 8.3**

Considerando os autômatos do exemplo 8.1 determine o autômato que reconhece a linguagem  $L(M_2).L(M_1)$ .

**8.3 Fecho de Kleene de autômatos finitos**

Seja  $M_1 = (A, Q_1, q_1, \delta_1, F_1)$  um autômato (AFD ou AFND) qualquer, e seja  $L(M_1)$  a linguagem por ele reconhecida. O AFND  $M = (A, Q, q_0, \delta, F)$ , onde

$$Q = Q_1 \cup \{q_0\}$$

$$F = \{q_0\}$$

$$\delta = \delta_1 \cup \{(q_0, \varepsilon, q_1)\} \cup (F_1 \times \{\varepsilon\} \times \{q_0\})$$

reconhece a linguagem  $L(M) = (L(M_1))^*$ , fecho de Kleene de  $M_1$ .

Na figura 8.3 ilustra-se graficamente a construção de  $M$  a partir de  $M_1$ . É claro da composição gráfica que  $M$  aceita  $\varepsilon$  ou qualquer palavra que percorra o caminho

$$q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{u_1} q_{f_1} \xrightarrow{\varepsilon} q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{u_2} \dots \xrightarrow{\varepsilon} q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{u_n} q_{f_n} \xrightarrow{\varepsilon} q_0, \quad \text{com } q_{f_i} \in F_1$$

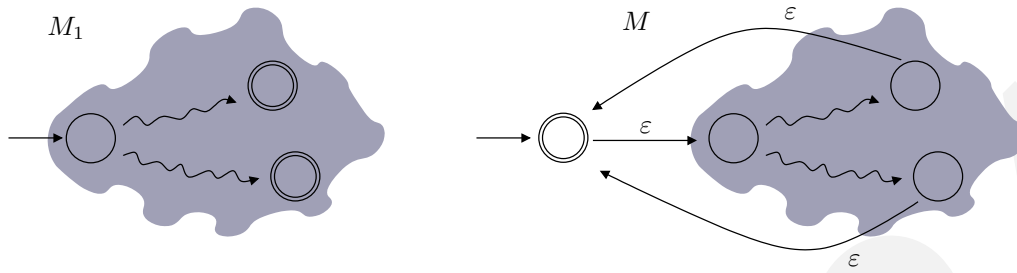


Figura 8.3: Ilustração gráfica do fecho de Kleene de autómatos finitos.

o que corresponde a zero ou mais concatenações de palavras de  $L(M_1)$ .

#### Exercício 8.4

Prove formalmente esta equivalência.

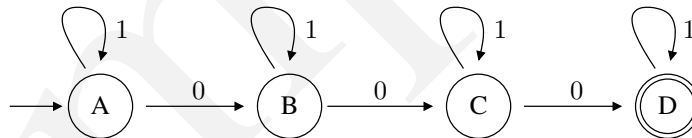
#### Exemplo 8.3

Sobre o alfabeto  $A = \{0, 1\}$  construa um autômato que reconhece a linguagem

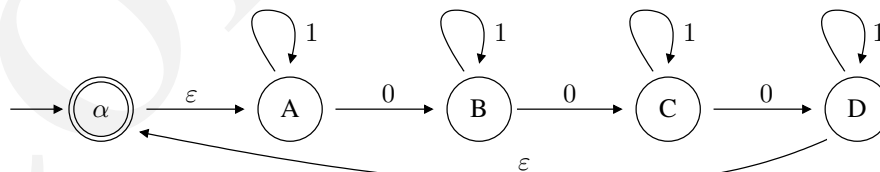
$$L = \{w \in A^* \mid \#(0, w) = 3\}$$

Com base nele construa o autômato que reconhece a linguagem  $L^*$ .

**Resposta:** A resposta à primeira questão é dada pelo autômato seguinte.

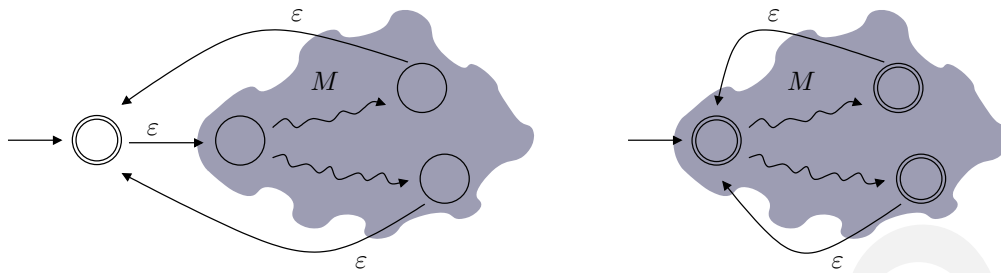


Aplicando-lhe o algoritmo de fecho obtém-se



#### Exercício 8.5

As duas construções da figura abaixo, definidas com base no autômato  $M$  (a parte a sombreado), parecem equivalentes, mas não o são. Apenas a da esquerda garante que o fecho de Kleene é bem construído sempre. Mostre que as duas construções só são equivalentes se o estado inicial de  $M$  for de aceitação ou se não houver transições a ele dirigidas.



- Considere que  $M$  é um autômato definido sobre o alfabeto  $A = \{a, b, c\}$ , reconhecendo as seqüências começadas por  $a$ . Mostre que as linguagens reconhecidas pelas duas construções são a mesma.
- Considere que  $M$  é um autômato definido sobre o alfabeto  $A = \{a, b, c\}$ , reconhecendo as seqüências terminadas por  $a$ . Mostre que as linguagens reconhecidas pelas duas construções são diferentes.

## 8.4 Complementação de autômatos finitos

Seja  $M_1 = (A, Q_1, q_1, \delta_1, F_1)$  um AFD (note, determinista) qualquer, e seja  $L(M_1)$  a linguagem por ele reconhecida. O AFD  $M = (A, Q, q_0, \delta, F)$  onde

$$Q = Q_1$$

$$q_0 = q_1$$

$$F = Q_1 - F_1$$

$$\delta = \delta_1$$

Se for AFND  
convertamos primeiro...

reconhece a linguagem  $L(M) = \overline{L(M_1)}$ , ou seja, a linguagem complementar de  $L(M_1)$  sobre o mesmo alfabeto. A única alteração efetuada é a complementação do conjunto de aceitação. Na realidade

$$u \in L(M) \Rightarrow \delta^*(q_0, u) \in F \Rightarrow \delta^*(q_0, u) \notin Q - F \Rightarrow u \notin L(M')$$

e

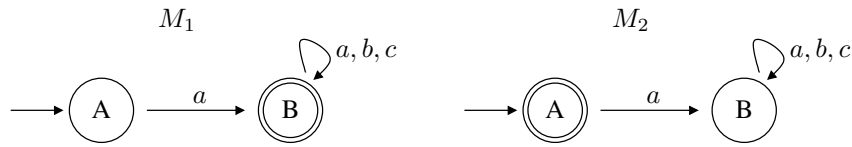
$$u \in L(M') \Rightarrow \delta^*(q_0, u) \in Q - F \Rightarrow \delta^*(q_0, u) \notin F \Rightarrow u \notin L(M)$$

A complementação dos estados de aceitação não funciona para AFND (autômatos não deterministas).

**Se se quiser complementar um AFND deve-se primeiro convertê-lo para um AFD equivalente e depois complementar este.**

### Exercício 8.6

Mostre que os dois autômatos da figura seguinte não são complementares um do outro. Para isso, basta apresentar uma palavra que ambos reconheçam ou ambos rejeitem. ("a")



Descreva a linguagem reconhecida pelo autômato da direita.  $L_2 = \{ \epsilon \}$

## 8.5 Intersecção de autômatos finitos

Dados dois autômatos  $M_1$  e  $M_2$  como construir um autômato  $M$  tal que  $L(M) = L(M_1) \cap L(M_2)$ , i.e., que a linguagem reconhecida por  $M$  seja a **intersecção** das linguagens reconhecidas por  $M_1$  e  $M_2$ ? Com base nas operações apresentadas anteriormente é possível chegar-se a tal autômato. Na realidade, por aplicação das leis de De Morgan,

$$L(M_1) \cap L(M_2) = \overline{\overline{L(M_1)} \cup \overline{L(M_2)}} \quad \text{Muito difícil...}$$

Logo, sabendo-se complementar e reunir autômatos finitos, sabe-se fazer a sua intersecção. Note que esta abordagem envolve 3 complementações, o que pode obrigar à conversão de 3 AFND para os AFD equivalentes.

Considere-se agora uma abordagem direta para a obtenção da intersecção de autômatos finitos. Uma palavra é reconhecida pela intersecção de dois autômatos se e só se for reconhecida por cada um dos dois autômatos individualmente. Isto permite construir a intersecção à volta do produto cartesiano dos conjuntos de estados dos dois autômatos. Sejam  $M_1 = (A, Q_1, q_1, \delta_1, F_1)$  e  $M_2 = (A, Q_2, q_2, \delta_2, F_2)$  dois autômatos (AFD ou AFND) quaisquer, e sejam  $L(M_1)$  e  $L(M_2)$  as linguagens por eles reconhecidas, respetivamente. O AFND  $M = (A, Q, q_0, \delta, F)$ , em que

$$Q = Q_1 \times Q_2$$

$$q_0 = (q_1, q_2)$$

$$F = F_1 \times F_2$$

$$\delta \subseteq (Q_1 \times Q_2) \times A_\epsilon \times (Q_1 \times Q_2)$$

*← não determinista!*

sendo  $\delta$  definido de modo que:

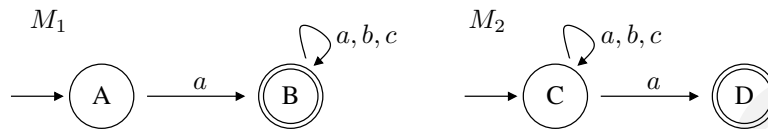
- $((q_i, q_j), a, (q'_i, q'_j)) \in \delta$  sse  $(q_i, a, q'_i) \in \delta_1$  e  $(q_j, a, q'_j) \in \delta_2$ , para todo o  $a \in A$ ;
- $((q_i, q_j), \epsilon, (q'_i, q'_j)) \in \delta$  sse  $(q_i, \epsilon, q'_i) \in \delta_1$ , para todo o  $q_j \in Q_2$ ;
- $((q_i, q_j), \epsilon, (q_i, q'_j)) \in \delta$  sse  $(q_j, \epsilon, q'_j) \in \delta_2$ , para todo o  $q_i \in Q_1$ ;
- $((q_i, q_j), \epsilon, (q'_i, q'_j)) \in \delta$  sse  $(q_i, \epsilon, q'_i) \in \delta_1$  e  $(q_j, \epsilon, q'_j) \in \delta_2$ ;

reconhece a linguagem  $L(M) = L(M_1) \cap L(M_2)$ , ou seja, a linguagem intersecção de  $L(M_1)$  e  $L(M_2)$ .



**Exemplo 8.4**

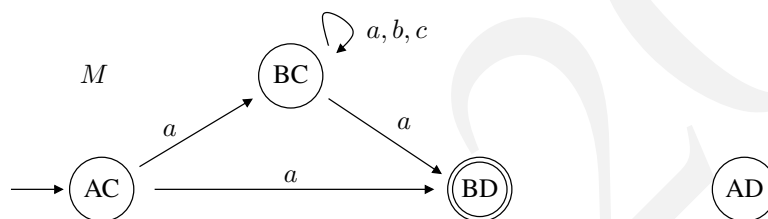
Sobre o alfabeto  $A = \{a, b, c\}$  considere os autómatos  $M_1$  e  $M_2$ , graficamente apresentados a seguir, que reconhecem respectivamente as linguagens das palavras começadas e terminadas por  $a$ .



Construa um autômato que reconheça a linguagem  $L = L(M_1) \cap L(M_2)$ .

**Resposta:**

A figura seguinte mostra o autômato obtido pela aplicação do mecanismo de construção definido acima.

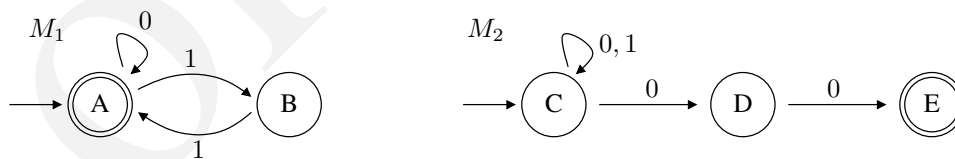


O estado  $AD$  não é alcançável a partir do estado inicial, pelo que pode ser eliminado da resposta, ficando-se com um autômato com 3 estados.

O exemplo anterior mostra que a aplicação do mecanismo de construção da intersecção de autómatos pode conduzir à introdução de estados não alcançáveis a partir do estado inicial. Isto propõe que a construção deve ser feita a partir do estado inicial, apenas se considerando os estados alcançáveis.

**Exemplo 8.5**

Sobre o alfabeto binário considere os autómatos  $M_1$  e  $M_2$ , graficamente apresentados a seguir

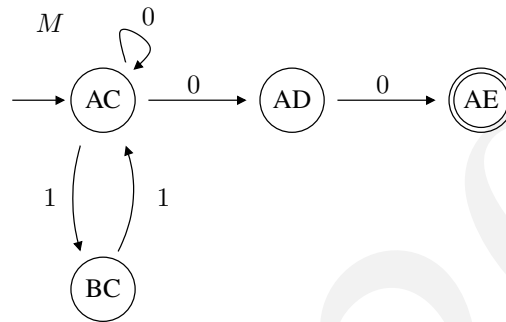


e determine a sua intersecção.

**Resposta:**

- O estado inicial do novo autômato é o estado  $AC = (A, C)$ . (Designar-se-á, no resto do exemplo, por  $XY$  o estado do autômato intersecção correspondente ao par  $(X, Y)$ .)
- Há uma transição a sair de  $A$  etiquetada com 0 — a transição  $(A, 0, A)$  — e duas a sair de  $C$  — as transições  $(C, 0, C)$  e  $(C, 0, D)$ . Logo, haverá duas transições a sair de  $AC$  etiquetadas com 0 — as transições  $(AC, 0, AC)$  e  $(AC, 0, AD)$ .

- Etiquetadas com 1 há uma transição a sair de  $A$ , dirigida a  $B$ , e uma a sair de  $C$ , dirigida a  $C$ , pelo que o autómato intersecção terá uma única seta etiquetada com 1 a sair de  $AC$  — a seta  $(AC, 1, BC)$ . Com estas setas foram alcançados os estados  $AD$  e  $BC$ .
- Procedendo da mesma forma com estes estados e com outros que, eventualmente, se alcançarem, obtém-se o autómato desenhado na figura seguinte, que possui apenas 4 estados e não os 6 do produto cartesiano.



É fácil constatar que  $M_1$  reconhece as sequências binárias com número par de uns e  $M_2$  as sequências binárias terminadas em 00.  $M$ , sendo a intersecção de  $M_1$  e  $M_2$ , reconhece as sequências binárias com um número par de uns terminadas em 00.

## 8.6 Diferença de autómatos

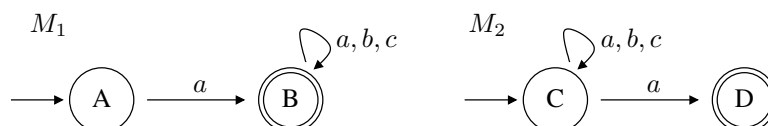
Dados dois autómatos  $M_1$  e  $M_2$  como construir um autómato  $M$  tal que  $L(M) = L(M_1) - L(M_2)$ , i.e., que a linguagem reconhecida por  $M$  seja a **diferença** entre linguagens reconhecidas por  $M_1$  e  $M_2$ ? Com base nas operações apresentadas anteriormente é possível chegar-se a tal autómato. Na realidade

$$L(M_1) - L(M_2) = L(M_1) \cap \overline{L(M_2)} = \overline{L(M_1)} \cup L(M_2)$$

Ora, já se mostrou como é que se constroem autómatos para a complementação e a intersecção ou para a reunião e a complementação.

### Exercício 8.7

Sobre o alfabeto  $A = \{a, b, c\}$  considere os autómatos  $M_1$  e  $M_2$ , graficamente apresentados a seguir, que reconhecem respetivamente as linguagens das palavras começadas e terminadas por  $a$ .



Construa um autómato que reconheça a linguagem  $L = L(M_1) - L(M_2)$ .

## Capítulo 9

# Equivalência entre Expressões Regulares e Autômatos Finitos

As expressões regulares e os autômatos finitos são equivalentes, no sentido em que descrevem a mesma classe de linguagens, as linguagens regulares. Assim sendo, é possível converter uma expressão regular dada num autômato finito que represente a mesma linguagem. Inversamente, é também possível converter um autômato finito dado numa expressão regular descrevendo a mesma linguagem.

### 9.1 Conversão de uma expressão regular num autômato finito

Como vimos na sua definição, uma expressão regular é contruída à custa de elementos primitivos e dos operadores escolha ( $|$ ), concatenação ( $.$ ) e fecho ( $*$ ). Os elementos primitivos correspondem à expressão  $\emptyset$ , representando o conjunto vazio, e às expressões do tipo  $a$  com  $a \in A$ , sendo  $A$  o alfabeto. É habitual considerar a expressão  $\varepsilon$  como um elemento primitivo, embora se saiba que  $\varepsilon = \emptyset^*$ .

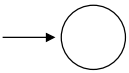
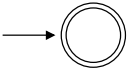
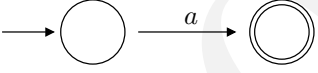
É possível decompor uma expressão regular num conjunto de elementos primitivos interligados pelos operadores referidos acima. Dada uma expressão regular qualquer ela é:

1. um elemento primitivo;
2. ou uma expressão do tipo  $e^*$ , sendo  $e$  uma expressão regular qualquer;
3. ou uma expressão do tipo  $e_1.e_2$ , sendo  $e_1$  e  $e_2$  duas expressões regulares quaisquer; ou
4. ou uma expressão do tipo  $e_1|e_2$ , sendo  $e_1$  e  $e_2$  duas expressões regulares quaisquer;

Se se identificar os autômatos equivalentes das expressões primitivas, tem-se o problema da conversão de uma expressão regular para um autômato finito resolvido, visto que se sabe como fazer a reunião, concatenação e fecho de autômatos (ver capítulo 8).

### 9.1.1 Autômatos dos elementos primitivos

A tabela seguinte mostra os autômatos correspondentes às expressões regulares  $\emptyset$ ,  $\varepsilon$  e  $a$ , com  $a \in A$ .

expressão regular	autômato finito
$\emptyset$	
$\varepsilon$	
$a$	

Na realidade, o autômato referente a  $\varepsilon$  pode ser obtido aplicando o fecho ao autômato de  $\emptyset$ .

### 9.1.2 Algoritmo de conversão

A transformação de uma expressão regular num autômato finito pode ser feita aplicando os seguintes passos:

1. Se a expressão regular é o do tipo primitivo, o autômato correspondente pode ser obtido da tabela anterior.
2. Se é do tipo  $e_1|e_2$ , aplica-se este mesmo algoritmo na obtenção de autômatos para as expressões  $e_1$  e  $e_2$  e, de seguida, aplica-se a reunião de autômatos descrita na secção 8.1.
3. Se é do tipo  $e^*$ , aplica-se este mesmo algoritmo na obtenção de um autômato equivalente à expressão regular  $e$  e, de seguida, aplica-se o fecho de autômatos descrita na secção 8.3.
4. Finalmente, se é do tipo  $e_1.e_2$ , aplica-se este mesmo algoritmo na obtenção de autômatos para as expressões  $e_1$  e  $e_2$  e, de seguida, aplica-se a concatenação de autômatos descrita na secção 8.2.

#### Exemplo 9.1

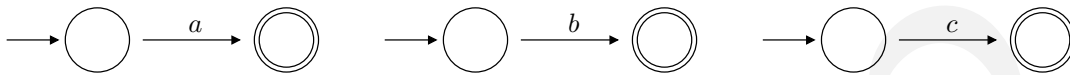
Construa um autômato equivalente à expressão regular  $e = a|a(a|b|c)^*a$ .

**Resposta:**

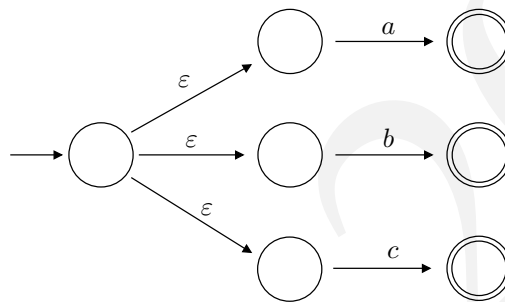
A expressão regular  $e$  é do tipo  $e_1|e_2$ , com  $e_1 = a$  e  $e_2 = a(a|b|c)^*a$ , pelo que se deve aplicar a regra 4 do algoritmo. A expressão  $e_1$  é do tipo primitivo, pelo que se pode aplicar a tabela para obter o autômato. A expressão  $e_2$  resulta da concatenação de 3 expressões regulares, para as quais temos de obter autômatos. Apenas a segunda expressão da concatenação tem de ser considerada, visto já se ter os autômatos das outras. Tem-se então que converter a expressão regular  $e_3 = (a|b|c)^*$ ,

que se obtém do fecho sobre o autômato que represente a expressão  $e_4 = a|b|c$ . Este, por sua vez, resulta da reunião dos autômatos das expressões primitivas  $a$ ,  $b$  e  $c$ .

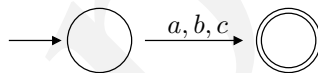
Uma vez definida a decomposição, a construção faz-se dos elementos primitivos para a expressão global. Os autômatos para as expressões  $a$ ,  $b$  e  $c$  são



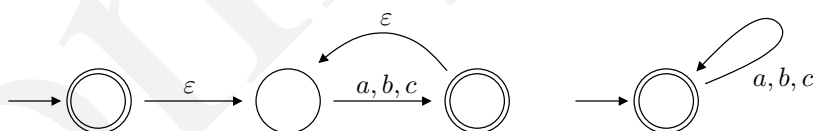
Fazendo a sua reunião obtém-se o autômato para  $e_4$



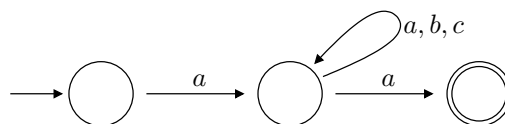
que pode ser simplificado para



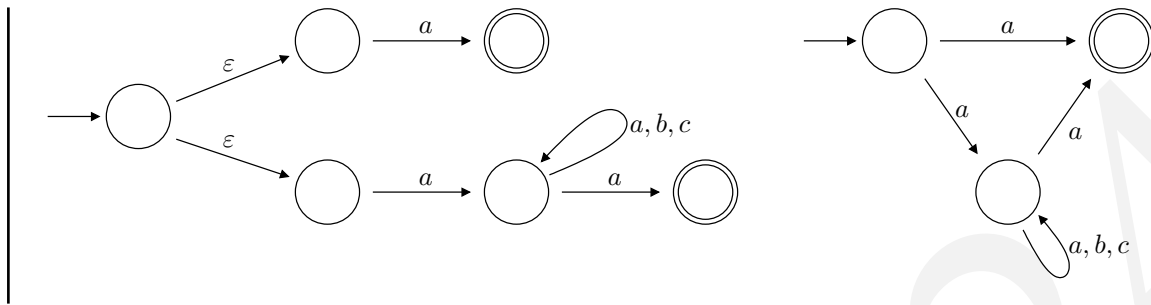
Aplicando o fecho a este autômato obtém-se o autômato para a expressão  $e_3$ , que se apresenta a seguir tal como resulta do fecho (lado esquerdo) e após simplificação (lado direito).



O autômato para  $e_2$  resulta da concatenação dos autômatos para  $a$ ,  $e_3$  e  $a$ , obtendo-se, após simplificação, o autômato seguinte



Finalmente, o autômato para  $e$  obtém-se da reunião dos autômatos para  $e_2$  e  $a$ , o que resulta no autômato seguinte, lado esquerdo (o lado direito representa o mesmo autômato após simplificação).



## 9.2 Conversão de um autômato finito numa expressão regular

### 9.2.1 Autômato finito generalizado

A conversão de autômatos finitos em expressões regulares baseia-se num novo tipo de autômatos, designados **autômatos finitos generalizados**. Este autômatos são semelhantes aos autômatos finitos não-deterministas mas em que as etiquetas dos arcos são expressões regulares.

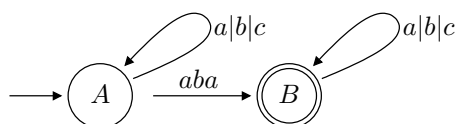
Um **autômato finito generalizado** (AFG) é um quintuplo  $M = (A, Q, q_0, \delta, F)$ , em que:

- $A$  é o alfabeto de entrada;
- $Q$  é um conjunto finito não vazio de estados;
- $q_0 \in Q$  é o estado inicial;
- $\delta \subseteq (Q \times \underline{E} \times Q)$  é a relação a transição entre estados, sendo  $E$  o conjunto das expressões regulares definidas sobre  $A$ ; e
- $F \subseteq Q$  é o conjunto dos estados de aceitação.

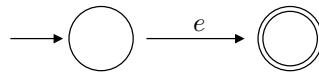
Note que com base nesta definição os autômatos finitos deterministas e não deterministas são autômatos finitos generalizados.

#### Exemplo 9.2

O autômato finito generalizado representado graficamente abaixo representa o conjunto das palavras, definidas sobre o alfabeto  $A = \{a, b, c\}$ , que contêm a sub-palavra  $aba$ .





Imagine que consegue transformar um autômato finito generalizado dado — pense, por exemplo, no autômato da figura anterior — num outro com a configuração dada pela figura seguinte



sendo  $e$  uma expressão regular qualquer. Então,  $e$  é uma expressão regular equivalente ao autômato dado. Um autômato como o da figura designa-se por **autômato finito generalizado reduzido**.

## 9.2.2 Algoritmo de conversão

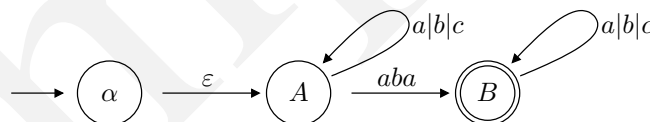
Assim sendo, a obtenção de uma expressão regular equivalente a um autômato qualquer dado, resume-se à transformação desse autômato num autômato finito generalizado reduzido. A redução de um AFG pode ser feita aplicando o seguinte procedimento:

1. transformação de um AFG noutro cujo estado inicial não tenha transições a ele dirigidas; 
2. transformação de um AFG noutro com um único estado de aceitação que não tenha transições que dele partam; 
3. eliminação, um a um, por transformações de equivalência, dos restantes estados.

Se o estado inicial de um AFG possui transições a ele dirigidas, a transformação definida pelo ponto 1 do procedimento anterior faz-se acrescentando um novo estado, que passa a ser o estado inicial do AFG transformado, e uma nova transição, etiquetada com  $\varepsilon$ , ligando este estado ao antigo estado inicial.

### Exemplo 9.3

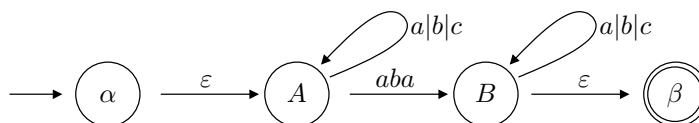
O AFG do exemplo 9.2 tem transições dirigidas ao seu estado inicial. O AFG representado abaixo, é-lhe equivalente e tal já não acontece.



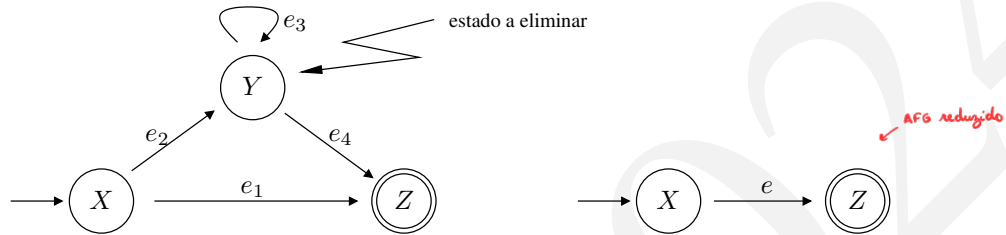
Se um AFG possui mais que um estado de aceitação ou se possui apenas um com transições que dele partem, a transformação definida pelo ponto 2 do procedimento de redução faz-se acrescentando um novo estado, que passa a ser o único estado de aceitação do AFG transformado, e novas transições etiquetadas com  $\varepsilon$  ligando os antigos estados de aceitação ao novo.

### Exemplo 9.4

O AFG do exemplo 9.3 tem um único estado de aceitação mas tem transições que dele partem. O AFG representado abaixo, é-lhe equivalente e tal já não acontece.



Está-se agora em condições de aplicar sucessivamente o ponto 3 do procedimento de redução, no sentido de eliminar os estados intermédios ( $A$  e  $B$ , no caso do exemplo). A ordem pela qual se deve proceder à sua eliminação é arbitrária, em termos da obtenção de uma expressão regular equivalente, mas pode afetar a complexidade das operações a realizar. A eliminação de estados preconizada pelo ponto 3 do procedimento de transformação está esquematizada na figura seguinte



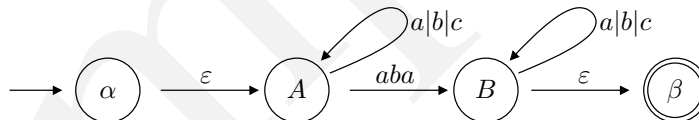
A situação inicial é representada pelo grafo da esquerda, sendo  $Y$  o estado que se pretende eliminar. É possível ir de  $X$  para  $Z$  passando por  $Y$ . Isso corresponde a sequências que encaixem na expressão regular  $e_2e_3^*e_4$ . Se se eliminar  $Y$ , estas sequências têm de ser colocadas diretamente entre  $X$  e  $Z$ . Se já existir uma transição entre  $X$  e  $Z$ , esta nova terá que ficar em paralelo. Chega-se assim à situação ilustrada à direita, sendo  $e$  dado por

$$e = e_1 | e_2e_3^*e_4$$

Note que, se  $e_1$  não existir, ficar-se-á apenas com  $e_2e_3^*e_4$ . Note ainda que, se  $e_3$  não existir,  $e_2e_3^*e_4 = e_2e_4$

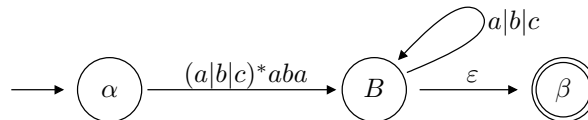
### Exemplo 9.5

Remova os estados  $A$  e  $B$  do AFG do exemplo 9.4, redesenhado aqui por comodidade

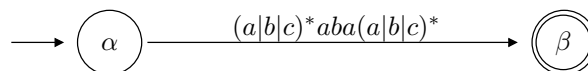


#### Resposta:

Comece-se por remover o estado  $A$ . Os estados  $\alpha$ ,  $A$  e  $B$  desempenham respetivamente os papéis dos estados  $X$ ,  $Y$  e  $Z$  no esquema relativo ao ponto 3 do procedimento de redução. Fazendo a correspondência, tem-se  $e_1 = \emptyset$ ,  $e_2 = \epsilon$ ,  $e_3 = a|b|c$  e  $e_4 = aba$ , pelo que o AFG após supressão do estado  $A$  se transforma no AFG seguinte, onde  $e = \emptyset | \epsilon(a|b|c)^*aba = (a|b|c)^*aba$ .



De seguida elimina-se o estado  $B$ . Neste caso os estados  $\alpha$ ,  $B$  e  $\beta$  desempenham respetivamente os papéis dos estados  $X$ ,  $Y$  e  $Z$ , resultando em





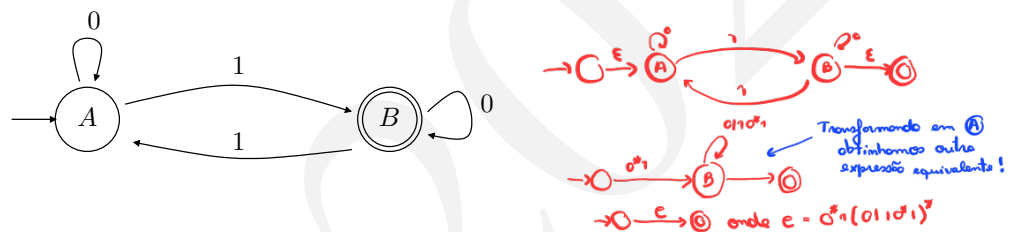
pelo que a expressão pretendida é

$$e = (a|b|c)^*aba(a|b|c)^*$$

O exemplo anterior não deixa transparecer alguma complexidade que pode aparecer no processo de remoção de estados. O estado a remover pode participar em vários triângulos  $X - Y - Z$ , o que obriga a calcular várias expressões regulares por cada estado a remover.

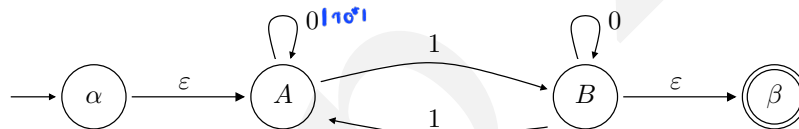
### Exemplo 9.6

Considere o autômato da figura seguinte e obtenha uma expressão regular equivalente



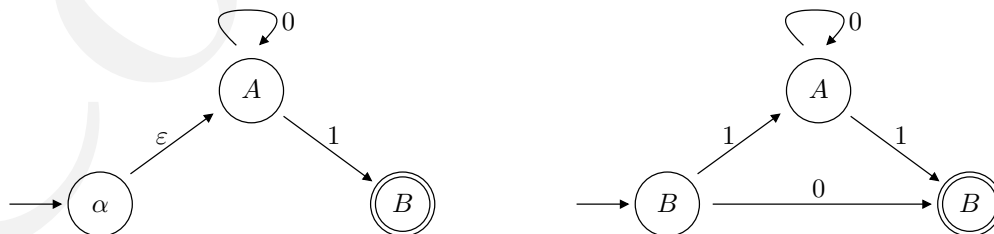
#### Resposta:

Comece-se por acrescentar novos estados inicial e de aceitação em consequência da aplicação dos pontos 1 e 2 do procedimento de redução. Obtém-se

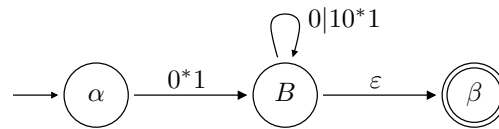


O estado  $A$  tem duas transições a si dirigidas, vindas de  $\alpha$  e  $B$ , e uma que dele parte, dirigida a  $B$ . A sua supressão vai obrigar ao acrescento de duas transições, uma de  $\alpha$  para  $B$  e outra de  $B$  para  $B$ . O estado  $B$  tem duas transições que dele saem, uma dirigida a  $A$  e outra a  $\beta$ , e uma a si dirigida, vinda de  $A$ . A sua supressão vai obrigar ao acrescento de duas transições, uma de  $A$  para  $\beta$  e outra de  $A$  para  $A$ . Sendo o custo de supressão de cada um dos dois estados igual, a ordem parece irrelevante. Opte-se então pela supressão de  $A$  em primeiro lugar.

Há dois triângulos  $X - Y - Z$  a considerar, um formado pelos estados  $\alpha$ ,  $A$  e  $B$  e outro pelos estados  $B$ ,  $A$  e  $B$ . Estes triângulos estão ilustrados na figura seguinte.



Da supressão de  $A$  resultam um arco entre os estados  $\alpha$  e  $B$  e outro de  $B$  para ele próprio. O primeiro terá a etiqueta  $e_1 = 0^*1$  e o segundo a etiqueta  $e_2 = 0|10^*1$ . Note que o arco original de  $B$  para  $B$  desaparece em consequência da supressão de  $A$ , sendo substituído pelo novo com etiqueta  $e_2$ . Após a supressão obtém-se o AFG seguinte



Suprimindo  $B$  obtém-se facilmente a expressão pretendida

$$e = 0^*1(0|10^*1)^*$$

### Exercício 9.1

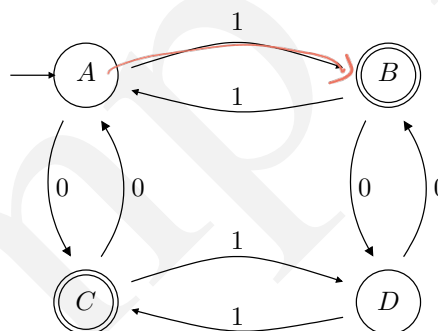
Repita o exercício do exemplo 9.6 mas eliminando em primeiro lugar o estado  $B$ .

**Resposta:**

Irá obter uma expressão regular diferente  $((0|10^*1)^*10^*)$ , mas que é equivalente à obtida no exemplo anterior.

### Exercício 9.2

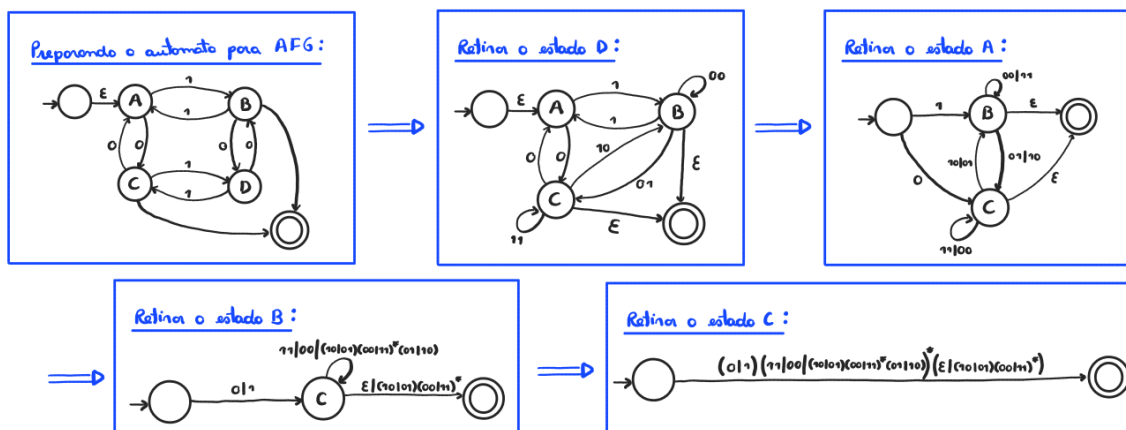
Determine uma expressão regular equivalente do autômato finito seguinte



*Este autômato serve para quê?*

→ Reconhece sequências com número ímpar de uns e par de zeros ou sequência com número ímpar de zeros e par de uns

*Nota: A ordem a retirar é irrelevante (mas pode ficar difícil...)*



Assim, analisando o AFG reduzido equivalente:

→  $\rightarrow \text{---} \xrightarrow{e} \text{---} \rightarrow$ , a expressão regular equivalente ao AF dado é:  $(0|1)(1|00|(10|1)(00|11)^*(01|10))^*(\epsilon|(10|1)(00|11)^*)$

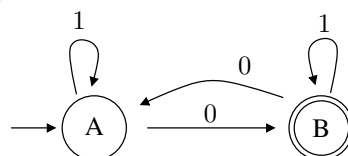
## Capítulo 10

# Equivalência entre Gramáticas Regulares e Autômatos Finitos

Se é possível converter uma ER  
e  $GR \leftrightarrow ER \dots$

As gramáticas regulares e os autômatos finitos são equivalentes, no sentido em que descrevem a mesma classe de linguagens, as linguagens regulares. Assim sendo, é possível converter-se uma gramática regular dada num autômato finito que represente a mesma linguagem. Inversamente, é também possível converter-se um autômato finito dado numa gramática regular que descreva a mesma linguagem.

Considere a figura seguinte, onde se representa um autômato finito (neste caso determinista) e uma gramática regular, ambos representando as sequências binárias com um número ímpar de uns.



A	→	0 B
		1 A
B	→	0 A
		1 B
		ε

A analogia entre ambos é óbvia, o que permite definir os algoritmos de conversão.

## 10.1 Conversão de AF em GR

### Algoritmo 10.1 (Conversão de AF em GR)

AFND-GR (input  $(A, Q, q_0, \delta, F)$ , output  $(T, N, P, S)$ ):

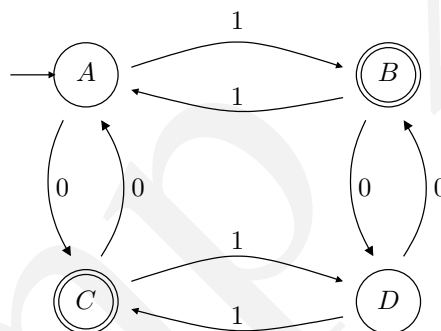
```

 $T \leftarrow A$ 
 $N \leftarrow Q$ 
 $S \leftarrow q_0$ 
 $P \leftarrow \{\}$ 
foreach  $(p, a, q) \in \delta$ 
     $P \leftarrow P \cup (p \rightarrow a q)$ 
foreach  $p \in F$ 
     $P \leftarrow P \cup (p \rightarrow \varepsilon)$ 

```

### Exemplo 10.1

Determine uma gramática regular equivalente do autômato finito seguinte



```

A → 1 B
  | 0 C
B → 1 A
  | 0 D
  | ε
C → 0 A
  | 1 D
  | ε
D → 1 C
  | 0 B

```

**Resposta:** Aplicando diretamente o algoritmo 10.1 e considerando que a gramática pretendida corresponde ao tuplo  $(T, N, P, S)$ , tem-se

```

 $T = \{0, 1\}$ 
 $N = \{A, B, C, D\}$ 
 $S = A$ 
 $P = \{A \rightarrow 0 C, A \rightarrow 1 B, B \rightarrow 0 D, B \rightarrow 1 A$ 
 $\quad C \rightarrow A, C \rightarrow 1 D, D \rightarrow 0 B, D \rightarrow 1 C$ 
 $\quad B \rightarrow \varepsilon, C \rightarrow \varepsilon\}$ 

```

Rescrevendo a gramática no formato mais habitual, tem-se

```

A → 0 C | 1 B
B → 0 D | 1 A | ε
C → 0 A | 1 D | ε
D → 0 B | 1 C

```

## 10.2 Conversão de GR em AF

Para a conversão de uma gramática regular num autômato finito, podem considerar-se duas situações:

- conversão para um autômato finito generalizado;
- conversão para um autômato finito não generalizado; *f Bosta converter...*

O primeiro caso é bastante simples e é dado pelo seguinte algoritmo.

### Algoritmo 10.2 (Conversão de GR em AFG)

GR→AFG (input  $(T, N, P, S)$ , output  $(A, Q, q_0, \delta, F)$ ):

$A \leftarrow T$

$Q \leftarrow N \cup \{q_f\}$ , com  $q_f \notin N$

$q_0 \leftarrow S$

$\delta \leftarrow \{\}$

**foreach**  $(X \rightarrow \omega Y) \in P$ , com  $\omega \in T^*$  and  $Y \in N$  // produções com não terminal no fim

$\delta \leftarrow \delta \cup (X, \omega, Y)$

**foreach**  $(X \rightarrow \omega) \in P$ , com  $\omega \in T^*$  // produções só com terminais (0 ou mais)

$\delta \leftarrow \delta \cup (X, \omega, q_f)$

$F \leftarrow \{q_f\}$  *Adiciona-se um novo estado, que vai ter  $\epsilon$ -transição*

O exemplo seguinte ilustra a aplicação deste algoritmo.

### Exemplo 10.2

Determine um autômato finito generalizado equivalente à gramática regular

$S \rightarrow a S \mid b S \mid c S \mid a b a X$

$X \rightarrow a X \mid b X \mid c X \mid \epsilon$

**Resposta:**

A aplicação direta do algoritmo 10.2 resulta no autômato  $M = (A, Q, q_0, \delta, F)$ , com

$A = \{a, b, c\}$

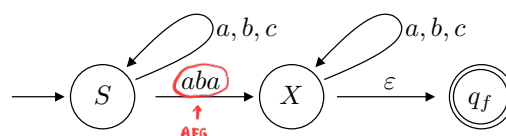
$Q = \{S, X, q_f\}$

$q_0 = S$

$\delta = \{(S, a, S), (S, b, S), (S, c, S), (S, aba, X),$   
 $(X, a, X), (X, b, X), (X, c, X), (X, \epsilon, q_f)\}$

$F = \{q_f\}$

Representado-o graficamente, tem-se



No caso de se pretender um autômato não generalizado como resultado, é preciso tratar adequadamente as produções dos tipos  $X \rightarrow \omega Y$  e  $X \rightarrow \omega$  quando  $\omega$  possui 2 ou mais símbolos terminais. É fácil perceber que se  $\omega = a_1 a_2$ , ou seja, se contiver 2 símbolos terminais, a produção anterior pode ser transformada em

$$X \rightarrow a_1 X_1$$

$$X_1 \rightarrow a_2 Y$$

em que  $X_1$  corresponde a um símbolo não terminal novo. Se  $\omega$  tiver mais que 2 símbolos terminais, o procedimento é semelhante. Se se transformar previamente a gramática de modo a que, em todas as produções,  $|\omega| = 1$ , o algoritmo anterior (algoritmo 10.2) conduz a um autômato finito não generalizado.

### Exemplo 10.3

Determine um autômato finito não generalizado equivalente à gramática regular

$$S \rightarrow a S \mid b S \mid c S \mid a b a X$$

$$X \rightarrow a X \mid b X \mid c X \mid \varepsilon$$

**Resposta:**

Comece-se por transformar a gramática de modo a quebrar a sequência  $aba$  na produção  $S \rightarrow a b a X$ , o que resulta em

$$S \rightarrow a S \mid b S \mid c S \mid a S_1$$

$$S_1 \rightarrow b S_2$$

$$S_2 \rightarrow a X$$

$$X \rightarrow a X \mid b X \mid c X \mid \varepsilon$$

A aplicação direta do algoritmo 10.2 a esta última gramática resulta no autômato  $M = (A, Q, q_0, \delta, F)$ , com

$$A = \{a, b, c\}$$

$$Q = \{S, S_1, S_2, X, q_f\}$$

$$q_0 = S$$

$$\delta = \{(S, a, S), (S, b, S), (S, c, S), (S, a, S_1), (S_1, b, S_2), (S_2, a, X), \\ (X, a, X), (X, b, X), (X, c, X), (X, \varepsilon, q_f)\}$$

$$F = \{q_f\}$$

Representado-o graficamente, tem-se

