Introdução às Linguagens Formais e Modelos de Computação

Ana Paula Tomás

Departamento de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto

Julho 2014

Conteúdo

Capítulo 1

Preliminares

Neste capítulo vamos recordar algumas noções elementares da teoria de conjuntos, grafos e relações binárias, e métodos de demonstração (por indução matemática e por redução ao absurdo).

1.1 Conjuntos

Tomamos a noção de **conjunto** como primitiva, dizendo que um *conjunto* é constituído por **elementos** que têm alguma propriedade em comum, que, no caso extremo, é o facto de pertencerem a esse conjunto. Os conjuntos podem ser **vazios** (sem elementos). Em geral, usamos letras maiúsculas para designar conjuntos e minúsculas para referir os seus elementos. Para indicar que a é um elemento do conjunto A, escrevemos $a \in A$. Um conjunto pode ser definido em *extensão*, exibindo todos os elementos que o constituem, ou *compreensão*, indicando a propriedade que caracteriza os seus elementos. Por exemplo, $\{1, 2, 3, 4\}$ e $\{n \mid n \in \mathbb{Z}^+ \text{ e } n \leq 4\}$ representam o mesmo conjunto em extensão e em compreensão. A tabela apresenta notação e nomenclatura habitual.

$a \in A$	a pertence a A, a é elemento de A
$a \notin A$	a não pertence a A
$A = B$	A igual a B (para todo x, tem-se $x \in A$ se e só se $x \in B$)
$A \subseteq B$	A contido em B, A subconjunto de B (para todo x, se $x \in A$ então $x \in B$)
$A \supseteq B$	A contém B, $B \subseteq A$
$A \subset B$	A contido propriamente em B, A subconjunto próprio de B, $A \subseteq B$ e $A \neq B$
$A \supset B$	A contém propriamente B, i.e., $B \subset A$
$A \neq B$	$A \not\subseteq B$ ou $B \not\subseteq A$
$\emptyset$ ou $\{\}$	conjunto vazio (i.e., sem qualquer elemento)

O **conjunto dos subconjuntos** de A representa-se por 2^A ou $\mathcal{P}(A)$. Qualquer conjunto A pertence ao conjunto dos seus subconjuntos, isto é, $A \in 2^A$. Por exemplo,

$$2^{\{1,2,3\}} = \{\emptyset,\ \{1\},\ \{2\},\ \{3\},\ \{1,2\},\ \{1,3\},\ \{2,3\},\ \{1,2,3\}\}$$

O conjunto $2^{\{1,2,3\}}$ tem 2^3 elementos. Se A tem n elementos então 2^A tem 2^n elementos, o que justifica a notação usada. Assim, o conjunto $2^{2^{\{1,2,3\}}}$ tem 2^{2^3} elementos (isto é, 256 elementos).

Um conjunto A não vazio é **finito** se e só se existir uma bijeção de A em $\{x \in \mathbb{N} \mid x < n\}$ para algum $n \in \mathbb{N}$. Nesse caso, n é o número de elementos de A e designa-se por **cardinal** de A. Usamos $|A|$ ou, alternativamente, $\#A$, para designar o cardinal de A. O cardinal do conjunto vazio é zero, isto é, $|\emptyset| = 0$.

Interpretação da notação. A mesma notação pode ter significados distintos, dependendo do contexto em que é usada. Por exemplo, quando x é um inteiro, é usual $|x|$ designar o valor absoluto de x (em particular, $|-5| = 5$). Nos capítulos seguintes, no contexto de linguagens formais, $|x|$ pode designar o número de símbolos que constituem uma palavra x, dando sentido a $|A| = 1$, $|-5| = 2$, $|\texttt{abbb}| = 4$ e $|-\texttt{6785}| = 5$, por exemplo. A propósito de questões de notação, é de salientar que em "$\{n \mid n \in \mathbb{N}\}$" e em "$\{n\}$, com $n \in \mathbb{N}$" estamos a referir conjuntos diferentes. O primeiro é $\mathbb{N}$ e o segundo é constituído por um único inteiro não negativo (que está a ser representado pela letra n).

Usaremos as designações habituais para os conjuntos dos naturais, inteiros, racionais e reais.

$\mathbb{N}$	naturais (inteiros não negativos)	$\mathbb{Z}_0^+$	inteiros não negativos
$\mathbb{Z}$	inteiros	$\mathbb{R}^-$	reais negativos
$\mathbb{Q}$	racionais	$\mathbb{R}^+$	reais positivos
$\mathbb{R}$	reais	$\mathbb{R}_0^+$	reais não negativos

$\mathbb{N} = \{0, 1, 2, 3, 4 \ldots\}$ é o conjunto dos inteiros não negativos (inclui 0). Não usamos a notação $\mathbb{N}_0$.

1.1.1 Operações com conjuntos

A **interseção de** A **e** B, representada por $A \cap B$, é o conjunto dos elementos que pertencem a ambos os conjuntos. A **união de** A **e** B, denotada por $A \cup B$, é o conjunto dos elementos que pertencem a pelo menos um dos conjuntos.

$$\begin{aligned} A \cap B &= \{x \mid x \in A \text{ e } x \in B\} = \{x \mid x \in A \wedge x \in B\} \\ A \cup B &= \{x \mid x \in A \text{ ou } x \in B\} = \{x \mid x \in A \vee x \in B\} \end{aligned}$$

O **complementar de** B **em** A (diferença entre A e B, ou A menos B), denotado por $A \setminus B$, é o conjunto dos elementos de A que não pertencem a B. Se estiver implicíto um *universo* $\mathcal{U}$, o complementar de A em $\mathcal{U}$ chama-se **complementar de** A e representa-se por $\overline{A}$.

$$\begin{aligned} A \setminus B &= \{x \mid x \in A \text{ e } x \notin B\} = \{x \mid x \in A \wedge x \notin B\} \\ \overline{A} = \mathcal{U} \setminus A &= \{x \mid x \in \mathcal{U} \text{ e } x \notin A\} \end{aligned}$$

Os exemplos seguintes ilustram a aplicação destas definições na demonstração de algumas propriedades.

Exemplo 1 Quaisquer que sejam os conjuntos $A, B \subseteq \mathcal{U}$, tem-se $\overline{A \cup B} = \overline{A} \cap \overline{B}$. De facto, se $x \in \overline{A \cup B}$ então, por definição de complementar, $x \notin A \cup B$. Logo, $x \notin A$ e $x \notin B$. Mas, $x \notin A$ sse $x \in \overline{A}$. E, $x \notin B$ sse $x \in \overline{B}$. Então, $x \in \overline{A}$ e $x \in \overline{B}$. Donde, $x \in \overline{A} \cap \overline{B}$. Mostrámos assim que $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$. Reciprocamente,

$$\begin{aligned} x \in \overline{A} \cap \overline{B} &\Rightarrow (x \in \overline{A} \wedge x \in \overline{B}) && \text{(por def. } \cap) \\ &\Rightarrow (x \notin A \wedge x \notin B) && \text{(por def. complementar)} \\ &\Rightarrow x \notin (A \cup B) && \text{(por def. } \cup) \\ &\Rightarrow x \in \overline{A \cup B} && \text{(por def. complementar)} \end{aligned}$$

ou seja, $\overline{A \cup B} \supseteq \overline{A} \cap \overline{B}$. Como mostrámos que $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$ e que $\overline{A \cup B} \supseteq \overline{A} \cap \overline{B}$, concluimos que $\overline{A \cup B} = \overline{A} \cap \overline{B}$, por definição de igualdade de conjuntos.

Exemplo 2 Vamos provar que $\overline{\overline{A} \cup \overline{B}} = A \cap B$. Pelo resultado provado no exemplo anterior, obtemos $\overline{\overline{A} \cup \overline{B}} = \overline{\overline{A}} \cap \overline{\overline{B}}$. Mas, $\overline{\overline{A}} \cap \overline{\overline{B}} = A \cap B$, pois o complementar do complementar de qualquer conjunto é ele próprio. Assim, $\overline{\overline{A}} = A$ e $\overline{\overline{B}} = B$. De facto, por definição de complementar, $x \in \overline{\overline{A}}$ sse $x \notin \overline{A}$. Mas, $x \notin A$ sse $x \in A$. Portanto, $\overline{\overline{A}} = A$, qualquer que seja A.

Exemplo 3 Vamos provar que $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$. Para tal, vamos provar que $x \in A \setminus (B \cup C)$ se e só se $x \in (A \setminus B) \cap (A \setminus C)$, qualquer que seja x.

$$\begin{aligned} x \in A \setminus (B \cup C) &\Leftrightarrow x \in A \wedge x \notin B \cup C && \text{(por def. de diferença)} \\ &\Leftrightarrow x \in A \wedge (x \notin B \wedge x \notin C) && \text{(por def. união)} \\ &\Leftrightarrow x \in A \setminus B \wedge x \in A \setminus C && \text{(por def. de diferença)} \\ &\Leftrightarrow x \in (A \setminus B) \cap (A \setminus C) && \text{(por def. interseção)} \end{aligned}$$

Convém relembrar as tabelas de verdade das conectivas lógicas do cálculo proposicional: conjunção ($\wedge$, "e"), disjunção ($\vee$, "ou"), negação ($\neg$), implicação ($\Rightarrow$, "se... então") e, ainda, equivalência ($\Leftrightarrow$, "se e só se") e ou-exclusivo ($\oplus$ ou $\dot{\vee}$, "ou... ou").

$\wedge$	V	F
V	V	F
F	F	F

$\vee$	V	F
V	V	V
F	V	F

$\neg$	V
V	F
F	V

$\Rightarrow$	V	F
V	V	F
F	V	V

$\Leftrightarrow$	V	F
V	V	F
F	F	V

$\oplus$	V	F
V	F	V
F	V	F

Sabemos que $(p \Rightarrow q)$ é equivalente a $(\neg p) \vee q$ e que $(p \Leftrightarrow q)$ é equivalente e $(p \Rightarrow q) \wedge (q \Rightarrow p)$.

Por vezes, usamos fórmulas baseadas na lógica de primeira ordem (lógica de predicados), com quantificação existencial ($\exists$, "existe") ou universal ($\forall$, "para todo") sobre variáveis num domínio de interpretação. Serão dados a seguir alguns exemplos.

Exemplo 4 Vamos analisar a veracidade ou falsidade das afirmações seguintes e justificá-la.

[V] Qualquer que seja $x \in \mathbb{Z}$, existe $y \in \mathbb{Z}$ tal que $x \leq y$ e $x \neq y$.

$$\forall x \in \mathbb{Z}\ \exists y \in \mathbb{Z}\ (x \leq y\ \wedge\ x \neq y)$$

A afirmação é verdadeira porque, sendo o conjunto dos inteiros infinito, se x é inteiro, $x + 1$ também é um inteiro e $x + 1$ é superior a x. Ou seja, dado x qualquer, se definirmos y como $x + 1$, satisfazemos a condição $(x \leq y\ \wedge\ x \neq y)$. Observemos que y depende naturalmente de x.

[F] Existe $y \in \mathbb{Z}$ tal que, para todo $x \in \mathbb{Z}$, se tem $x \leq y$.

$$\exists y \in \mathbb{Z}\ \forall x \in \mathbb{Z}\ x \leq y$$

A afirmação é falsa porque se x fosse $y + 1$, teriamos que ter $y + 1 \leq y$, o que não é satisfazível (pois é equivalente a $1 \leq 0$). Contrariamente ao caso anterior, agora y não depende de x, tornando a proposição falsa.

[V] Existe um inteiro não negativo que não excede qualquer outro inteiro não negativo.

$$\exists x \in \mathbb{Z}_0^+\ \forall y \in \mathbb{Z}_0^+\ x \leq y$$

A afirmação é verdadeira. O inteiro 0 (zero) pertence a $\mathbb{Z}_0^+$ e é menor ou igual que cada um dos inteiros não negativos. Aqui, $\mathbb{Z}_0^+$ denota o conjunto dos inteiros não negativos, o qual identificámos também por $\mathbb{N}$.

[F] Existe $x \in \mathbb{Z}$ tal que x é maior do que qualquer outro inteiro y.

$$\exists x \in \mathbb{Z}\ \forall y \in \mathbb{Z}\ x > y$$

A afirmação é falsa. O argumento pode ser análogo ao que usámos para mostrar a falsidade da segunda afirmação. Supondo que x existia, então, para $y = x + 1$, a condição $x > y$ também seria satisfeita. Mas, tal é absurdo, pois $x \not> x + 1$. Portanto, x não pode existir.

[F] Para todo $A \subseteq \mathbb{Z}$, tem-se $2^A = \{\emptyset\}$.

A afirmação é obviamente falsa, porque $\{1\}$ é um subconjunto de $\mathbb{Z}$ e $2^{\{1\}} = \{\emptyset, \{1\}\} \neq \{\emptyset\}$.

[V] Para todo $A \subseteq \mathbb{Z}$, se $A = \emptyset$ então $2^A = \{\emptyset\}$.

A afirmação é verdadeira. Só existe um subconjunto de $\mathbb{Z}$ que é vazio (o conjunto $\emptyset$), e $2^\emptyset = \{\emptyset\}$.
A propósito, notemos que, $\emptyset = \{\,\} \neq \{\emptyset\}$, ou seja, $\{\emptyset\}$ é um conjunto que tem um elemento. Esse elemento é $\emptyset$ e, como seria de esperar, o número de elementos de $2^\emptyset$ é $2^0 = 1$.

[F] Tem-se $2^A = \emptyset$, para algum $A \subseteq \mathbb{Z}$.

$$\exists A \;\; (A \subseteq \mathbb{Z} \;\wedge\; 2^A = \emptyset)$$

A afirmação é falsa, porque qualquer que seja o subconjunto A de $\mathbb{Z}$, o conjunto vazio é um elemento de 2^A.

[V] Quaisquer que sejam $x, y \in \mathbb{Z}$, tem-se $x \leq y$ ou $y \leq x$.

$$\forall x \in \mathbb{Z}\; \forall y \in \mathbb{Z}\; (x \leq y \;\vee\; y \leq x)$$

Dizer que $x \leq y$ equivale a dizer que existe um inteiro não negativo z tal que $y = x + z$. Assim, é verdade que $(x \leq y \;\vee\; y \leq x)$, quaisquer que sejam os inteiros x e y. De facto, como $x - y$ é sempre inteiro se x e y forem inteiros, podemos concluir que, se $x - y$ é não negativo, então $y \leq x$ pois $x = y + (x - y)$. Se $x - y$ é negativo, então $y - x$ é um inteiro positivo, e como $y = x + (y - x)$, tem-se $x \leq y$.

[F] Qualquer que seja $A \subseteq \mathbb{Z}$, se $A \neq \{-1, 2, 3\}$ então $4 \in A$.

$$\forall A \;\; (\,(A \subseteq \mathbb{Z} \;\wedge\; A \neq \{-1, 2, 3\}) \Rightarrow 4 \in A)\,)$$

Falso. Existe um subconjunto de $\mathbb{Z}$ que é diferente de $\{-1, 2, 3\}$ e que não tem 4. Por exemplo, $A = \emptyset$.

[V] Quaisquer que sejam $A, B \subseteq \mathbb{Z}$, se $5 \in A \setminus B$ então $5 \in A$.

$$\forall A \;\; \forall B(\,(A \subseteq \mathbb{Z} \;\wedge\; B \subseteq \mathbb{Z} \;\wedge\; 5 \in A \setminus B) \Rightarrow 5 \in A\,)$$

A afirmação é verdadeira. Quaisquer que sejam os subconjuntos A e B de $\mathbb{Z}$, tem-se $5 \in A \setminus B$ se e só se $5 \in A$ e $5 \notin B$. Logo, se $5 \in A \setminus B$ então $5 \in A$.

[V] Para todo $x \in \mathbb{Z}$ e quaisquer subconjuntos $A, B \subseteq \mathbb{Z}$, se $x \in A \setminus B$ então $x \in A$.

A afirmação é verdadeira. A justificação é semelhante à dada para a afirmação anterior (claro que é necessário referir x e não 5). Por definição, $x \in A \setminus B$ se e só se $x \in A$ e $x \notin B$. Logo, se $x \in A \setminus B$ então $x \in A$.

[F] Existe $x \in \mathbb{Z}$ tal que $x \in A \setminus B$, quaisquer que sejam $A, B \subseteq \mathbb{Z}$.

Esta afirmação pode ser traduzida pela fórmula lógica

$$\exists x \, (x \in \mathbb{Z} \ \wedge (\ \forall A \, \forall B \ (\ (B \subseteq \mathbb{Z} \wedge A \subseteq \mathbb{Z}) \Rightarrow x \in A \setminus B) \ ))$$

a qual, por vezes, escrevemos como $\exists x \in \mathbb{Z} \ \forall A \subseteq \mathbb{Z} \forall B \subseteq \mathbb{Z} \ \ (x \in A \setminus B)$. A afirmação é falsa. Sabemos que, quaisquer que sejam os conjuntos A e B, se $A = B$ então $A \setminus B = \emptyset$. Assim, se tomarmos, por exemplo, $A = B = \{1\}$, os conjuntos A e B são subconjuntos de $\mathbb{Z}$ e não existe qualquer inteiro x tal que $x \in A \setminus B$.

[V] $2 + 2 = 4$ ou $\sqrt{2} \in \mathbb{Z}$.

A afirmação é verdadeira porque, embora $\sqrt{2} \notin \mathbb{Z}$, é verdade que $2 + 2 = 4$.

[V] Se $2 + 2 \neq 4$ então $\sqrt{2} \in \mathbb{Z}$.

A afirmação é equivalente a "$2 + 2 = 4$ ou $\sqrt{2} \in \mathbb{Z}$" (ou seja, à anterior).

[V] Se $\sqrt{2} \in \mathbb{Z}$ então $2 + 2 \neq 4$.

A afirmação é equivalente a "$\sqrt{2} \notin \mathbb{Z}$ ou $2 + 2 \neq 4$", e como $\sqrt{2} \notin \mathbb{Z}$, a afirmação é verdadeira.

[V] $\sqrt{2} \notin \mathbb{Z}$ ou $2 + 2 \neq 4$.

A afirmação é verdadeira porque $\sqrt{2} \notin \mathbb{Z}$.

Exercício 1.1.1 Mostrar as propriedades seguintes, supondo que os conjuntos A, B e C indicados são subconjuntos quaisquer de um mesmo universo $\mathcal{U}$. A abreviatura "sse" significa "se e só se".

(a) $A \setminus B = A \cap \overline{B} = \overline{B} \setminus \overline{A}$

(b) $A \setminus B = \emptyset$ sse $A \subseteq B$

(c) $A \setminus B = A$ sse $A \cap B = \emptyset$

(d) $(A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (B \cap A)$

(e) $(A \cap B) \cap C = A \cap (B \cap C)$

(f) $(A \cup B) \cup C = A \cup (B \cup C)$

(g) $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$

(h) $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$

(i) $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$

(j) $\overline{\emptyset} = \mathcal{U}$

(k) $A \setminus A = \emptyset$

(l) $\overline{\overline{A}} = A$

(m) $A \cup \overline{A} = \mathcal{U}$

(n) $A \cap \mathcal{U} = A$

(o) $A \cup \emptyset = A$

(p) $\overline{A \cup B} = \overline{A} \cap \overline{B}$

(q) $A \subseteq B$ sse $\overline{B} \subseteq \overline{A}$

(r) $\overline{\mathcal{U}} = \emptyset$

(s) $A \setminus \emptyset = A$

(t) $A \cup \mathcal{U} = \mathcal{U}$

(u) $A \cap \overline{A} = \emptyset$

(v) $A \cap A = A$

1.2 Indução Matemática

O método de demonstração por indução matemática (ou indução finita) será bastante usado. Vamos recordá-lo.

Exemplo 5 Imaginemos *uma escada com uma infinidade de degraus*. Esta escada não é finita, pois tem sempre um degrau acima de qualquer outro que se considere. Suponhamos que é verdade (1.1).

"Se conseguir chegar até um degrau (qualquer), então também consigo chegar ao seguinte." (1.1)

Se nada mais for dito, não podemos concluir que "conseguimos chegar ao $105^{\underline{o}}$ degrau".

Suponhamos agora que não somente (1.1) mas também (1.2) se verifica.

"Conseguimos chegar ao $13^{\underline{o}}$ degrau." (1.2)

O que podemos concluir? Como conseguimos chegar ao $13^{\underline{o}}$ e é verdade (1.1), então conseguimos chegar ao $14^{\underline{o}}$. Como conseguimos chegar ao $14^{\underline{o}}$ e é verdade (1.1), então conseguimos chegar ao $15^{\underline{o}}$. Como conseguimos chegar ao $15^{\underline{o}}$ e é verdade (1.1), então conseguimos chegar ao $16^{\underline{o}}$, e sucessivamente. Para cada valor n, com $n \geq 13$, podemos construir a prova de que conseguimos chegar ao degrau n. Em suma, de (1.1) e (1.2), conclui-se (1.3).

"Conseguimos chegar ao n-ésimo degrau, qualquer que seja $n \geq 13$." (1.3)

Do mesmo modo, se for verdade que "Conseguimos chegar ao $1^{\underline{o}}$ degrau" e que "Se conseguirmos chegar até um degrau (qualquer), então também conseguimos chegar ao seguinte", podemos concluir que "Conseguimos chegar ao n-ésimo degrau, qualquer que seja $n \geq 1$".

O princípio de indução irá permitir traduzir formalmente provas deste tipo, como se ilustra a seguir.

Exemplo 6 Seja A_n a área de um quadrado de lado 2^n, com $n \geq 1$ (inteiro). Vamos mostrar que o resto da divisão de A_n por 3 é 1, qualquer que seja $n \geq 1$.

1. (Caso de base) Mostremos que o resto da divisão de A_1 por 3 é 1.

   Prova: É trivial, pois $A_1 = 4$ e sabemos que $4 = 3 \times 1 + 1$.

2. (Hereditariedade) Mostremos que, qualquer que seja $k \geq 1$, **se** A_k for da forma $3p + 1$, para algum $p \in \mathbb{Z}_0^+$, **então** $A_{k+1} = 3q + 1$ para algum $q \in \mathbb{Z}_0^+$.

   Prova: Por definição de A_{k+1} tem-se $A_{k+1} = (2^{k+1})^2 = 4(2^k)^2 = 4A_k$. Logo, se $A_k = 3p + 1$ (por hipótese) então $A_{k+1} = 4(3p + 1) = 3(4p + 1) + 1 = 3q + 1$, com $q = 4p + 1$. Portanto, se $A_k = 3p + 1$, para algum $p \in \mathbb{Z}_0^+$, então $A_{k+1} = 3q + 1$, para algum $q \in \mathbb{Z}_0^+$, já que se $p \in \mathbb{Z}_0^+$ então $4p + 1 \in \mathbb{Z}_0^+$.

De (1) e (2), o que podemos concluir sobre A_1, A_2, A_3,...?

Por (1), sabemos que o resto da divisão de A_1 por 3 é 1.
Como o resto da divisão de A_1 por 3 é 1 e mostrámos (2), concluimos que o resto da divisão de A_2 por 3 é 1.
Como o resto da divisão de A_2 por 3 é 1 e mostrámos (2), concluimos que o resto da divisão de A_3 por 3 é 1.
Como o resto da divisão de A_3 por 3 é 1 e mostrámos (2), concluimos que o resto da divisão de A_4 por 3 é 1.
$\vdots$

Do mesmo modo que conseguimos mostrar que A_4 dá resto 1 quando dividido por 3, podemos apresentar a dedução de que A_n dá resto 1 quando dividido por 3, para cada $n \geq 1$ que for dado. Assim, de (1) e (2), podemos concluir que, qualquer que seja $n \geq 1$, a área do quadrado de lado 2^n excede numa unidade um múltiplo de 3. □

1.2.1 Indução fraca

Proposição 1 *(**Princípio de Indução Matemática**) Escrevamos $P(n)$ como abreviatura de "o inteiro não negativo n satisfaz a propriedade P". Tem-se $\forall_{n\in\mathbb{N}}\ P(n)$, se forem satisfeitas as duas condições (i) e (ii) seguintes:*

(i) $P(0)$ *é verdade;*

(ii) $\forall k \in \mathbb{N}\ (\ P(k) \Rightarrow P(k+1)\ )$ *é verdade, isto é, para todo $k \in \mathbb{N}$, se $P(k)$ então $P(k+1)$.*

Aqui, $P(0)$ diz-se (caso de) **base** de indução. Em $P(k) \Rightarrow P(k+1)$, chamamos a $P(k)$ a **hipótese** de indução e a $P(k+1)$ a **tese**. A condição (ii) traduz a **hereditariedade** da propriedade.

Proposição 2 *(**Versão geral do Princípio de Indução**)*

Seja $n_0 \in \mathbb{Z}$ e suponhamos que $P(n)$ denota "o inteiro n satisfaz a propriedade P. Se forem satisfeitas as duas condições (i) e (ii) seguintes, então, para todo o inteiro $n \geq n_0$ tem-se $P(n)$.

(i) $P(n_0)$ *é verdade;*

(ii) $\forall k \geq n_0\ (\ P(k) \Rightarrow P(k+1)\ )$ *é verdade, isto é, para todo inteiro $k \geq n_0$, se $P(k)$ então $P(k+1)$.*

Exemplo 7 Provar que $(a+b)^n < 2^n(a^n + b^n)$, quaisquer que sejam a e b, reais positivos, e $n \geq 1$ inteiro.

Para efetuar a *prova por indução*, temos que mostrar que as duas condições de aplicabilidade do princípio de indução matemática se verificam neste caso.

- (Caso de base) Para $n = 1$, temos $(a+b)^1 = a + b < 2(a+b) = 2^1(a^1 + b^1)$, porque $a, b \in \mathbb{R}^+$.

- (Hereditariedade) Suponhamos que, para um dado $k \geq 1$, fixo mas arbitrário, se tem $(a+b)^k < 2^k(a^k + b^k)$, para $a, b \in \mathbb{R}^+$ quaisquer, e vamos mostrar que então $(a+b)^{k+1} < 2^{k+1}(a^{k+1} + b^{k+1})$.

  Como $(a+b)^{k+1} = (a+b)^k(a+b)$ e, por hipótese, $(a+b)^k < 2^k(a^k + b^k)$, concluimos que

  $$(a+b)^{k+1} < 2^k(a^k + b^k)(a+b) = 2^k(a^{k+1} + b^{k+1} + a^k b + b^k a)$$

  e como

  $$2^k(a^{k+1} + b^{k+1} + a^k b + b^k a) = 2^k(a^{k+1} + b^{k+1}) + 2^k(a^k b + b^k a),$$

  vamos mostrar que

  $$a^k b + b^k a \leq a^{k+1} + b^{k+1}$$

  ou seja, que $a^k b + b^k a - (a^{k+1} + b^{k+1}) \leq 0$, para poder concluir que $(a+b)^{k+1} < 2^{k+1}(a^{k+1} + b^{k+1})$, como pretendemos, para a, b reais positivos quaisquer. Ora, sabemos que

  $$a^k b + b^k a - a^{k+1} - b^{k+1} = -(a^k - b^k)(a - b) \leq 0,$$

  já que quaisquer que sejam $x, y \in \mathbb{R}^+$, se $x \leq y$ então $x^p \leq y^p$, para todo $p \geq 1$. Basta relembrarmos que as funções $f_p(x) = x^p$, são estritamente crescentes em $\mathbb{R}^+$. □

Exemplo 8 Vamos provar, por indução matemática, que $4^n + 15n - 1$ é múltiplo de 9, para todo $n \geq 1$.

Prova: Usando o princípio de indução, podemos concluir que a condição se verifica para todo $n \geq 1$ se mostrarmos que se verifica para $n = 1$ e que se se verificar para um certo valor de n, com $n \geq 1$ fixo mas arbitrário, então também se verifica para $n + 1$. Ora, para $n = 1$, tem-se $4^n + 15n - 1 = 4 + 15 - 1 = 18 = 2 \times 9$. Portanto, o caso de base verifica-se. Resta mostrar que a propriedade é hereditária. Para tal, observemos que $4^{n+1} + 15(n+1) - 1 = 4(4^n + 15n - 1) - (60n - 4) + 15(n+1) - 1 = 4(4^n + 15n - 1) - 45n + 18$. Assim, é óbvio que, se $4^n + 15n - 1$ for múltiplo de 9, então $4^{n+1} + 15(n+1) - 1$ também é múltiplo de 9, pois é soma algébrica de múltiplos de 9. □

1.2.2 Indução forte

Em certos casos é útil reforçar a hipótese de indução.

Exemplo 9 Para provar que "qualquer inteiro maior do que 1 é ou primo ou produto de primos" é necessário reforçar a hipótese de indução pois não basta ter a decomposição de k em primos para construir a decomposição de $k + 1$. Por exemplo, a decomposição de 12 não depende da decomposição de 11, mas sim das decomposições de 3 e 4 (ou, equivalentemente, das de 2 e 6). Assim, para k fixo, é conveniente assumir que, já se provou que i é primo ou produto de primos, para todo $i \in \mathbb{N}$ tal que $2 \leq i \leq k$. A prova por indução matemática ficaria então:

- (Caso de base) Pela definição de primo, 2 é primo (portanto, 2 é primo ou produto de primos).
- (Hereditariedade) Para $k \geq 2$ fixo, suponhamos que já mostrámos que todo i, tal que $2 \leq i \leq k$, é ou primo ou produto de primos. Vamos mostrar que então também podemos concluir que $k+1$ é primo ou produto de primos. De facto, se $k+1$ não for primo, existem $i_1, i_2 \in \mathbb{N} \setminus \{0,1\}$ tais que $k+1 = i_1 i_2$. Como $2 \leq i_1 \leq k$ e $2 \leq i_2 \leq k$, sabemos já que i_1 é primo ou produto de primos e i_2 é primo ou produto de primos.

  Analisando os quatro casos (a) i_1 e i_2 primos, (b) i_1 e i_2 produtos de primos, (c) i_1 primo e i_2 produto de primos, e (d) i_1 produto de primos e i_2 primo, e tomando tais decomposições de i_1 e i_2, concluimos que se $k+1 = i_1 i_2$ então $k+1$ pode-se reescrever como produto de primos. Logo, se $k+1$ não é primo, $k+1$ é um produto de primos, ou seja, $k+1$ é primo ou produto de primos.

Mostrámos que 2 é primo (ou produto de primos) e que, qualquer que seja $k \geq 2$, se todo i tal que $2 \leq i \leq k$ for primo ou produto de primos então $k+1$ é primo ou produto de primos. Portanto, por indução, k é primo ou produto de primos, qualquer que seja $k \geq 2$. □

Esta prova é suportada pela seguinte versão do Princípio de indução.

Proposição 3 (Versão forte do Princípio de indução) *Seja $P(n)$ uma condição na variável $n \in \mathbb{Z}$. Dado $n_0 \in \mathbb{Z}$, se forem satisfeitas as duas condições (i) e (ii) seguintes, então $\forall n \geq n_0 \;\; P(n)$.*

(i) *$P(n_0)$ é verdade;*

(ii) *para todo $k \in \mathbb{Z}$, se se tem $P(i)$ para todo $i \in \mathbb{Z}$, com $n_0 \leq i \leq k$, então tem-se $P(k+1)$.*

Proposição 4 *Os princípios de indução forte e fraca são equivalentes: se existir uma prova de $\forall n \in \mathbb{N} \;\; P(n)$ por indução fraca então também existe uma prova por indução forte e, reciprocamente, se existir uma prova por indução forte, existe uma prova por indução fraca.*

Prova: É trivial que, se existir uma prova de $\forall n \in \mathbb{N} \;\; P(n)$ por indução fraca então existe uma prova por indução forte, pois a mesma prova serve! Portanto, resta-nos mostrar que se existir uma prova por indução forte então existe uma prova por indução fraca. Para tal, vamos mostrar que se existe uma prova de $\forall n \in \mathbb{N} \;\; P(n)$ por indução forte, então existe uma prova por indução *fraca* de $\forall n \in \mathbb{N} \;\; (\forall i \leq n \;\; P(i))$. Podemos depois concluir que existe uma prova por indução fraca de $\forall n \in \mathbb{N} \; P(n)$ porque $(\forall n \in \mathbb{N} \;\; (\forall i \leq n \;\; P(i))) \Rightarrow \forall n \in \mathbb{N} \; P(n)$.

Seja $Q(n)$ a condição $\forall i \leq n \;\; P(i)$.

Mostrar que $Q(0)$ é verdade, é mostrar que $\forall i \leq 0 \;\; P(i)$, ou seja, que $P(0)$ é verdade. Como se teve de mostrar que $P(0)$ era verdade para provar $\forall n \; P(n)$ por indução forte, basta-nos reproduzir essa justificação.

Para mostrar que $\forall n \ \ (\ Q(n) \Rightarrow Q(n+1) \ )$, ou seja, que $\forall n \ \ (\ (\forall i \leq n \ \ P(i)) \Rightarrow (\forall i \leq n+1 \ \ P(i)) \ )$, é útil recordar que existe uma prova de $\forall n \ \ (\ (\forall i \leq n \ \ P(i)) \Rightarrow P(n+1) \ )$, que teve de ser feita para mostrar $\forall n \ P(n)$ por indução forte. E, como $\forall n \ ((\forall i \leq n \ P(i)) \Rightarrow (\forall i \leq n \ P(i)))$, tem-se

$$\forall n \ ((\forall i \leq n \ P(i)) \Rightarrow (P(n+1) \wedge (\forall i \leq n \ P(i))))$$

ou seja, $\forall n \in \mathbb{N} \ (Q(n) \Rightarrow Q(n+1))$. Assim, mostrámos que $Q(0)$ era verdade e $\forall n \in \mathbb{N} \ (Q(n) \Rightarrow Q(n+1))$. Logo, pelo princípio de indução fraca, concluimos $\forall n \in \mathbb{N} \ Q(n)$. □

1.2.3 Provas erradas

É necessário mostrar as duas condições de aplicabilidade do princípio de indução, sem o que as provas não ficam completas e, no pior dos casos, os resultados mal "demonstrados" são mesmo falsos.

Exemplo 10 Analisemos a seguinte demonstração (**claramente, errada!**) de que "entre dois inteiros consecutivos existe uma infinidade de inteiros".

> *Prova: Para todo $x \in \mathbb{R}$ (em particular para x inteiro) se $k \leq x \leq k+1$ então $k+1 \leq x+1 \leq k+2$, qualquer que seja $k \in \mathbb{N}$. Assim, se entre k e $k+1$ existir uma infinidade de inteiros, então entre $k+1$ e $k+2$ também existe uma infinidade de inteiros. De facto, a cada inteiro x no intervalo $[k, k+1]$ podemos associar um inteiro x' no intervalo $[k+1, k+2]$, a saber, por exemplo x' é $x+1$.*
>
> *Logo, por indução matemática sobre k, concluimos que entre dois inteiros consecutivos existe uma infinidade de inteiros. (c.q.d)*

Como entre dois inteiros consecutivos não há nenhum inteiro, a prova está errada. Onde está o erro? Não se provou que existia um intervalo $[k, k+1]$ com uma infinidade de inteiros. Apenas se mostrou a condição (ii) do princípio de indução, isto é, que a propriedade era *hereditária*. A condição (i) não foi provada, nem se pode provar!

Exemplo 11 A demonstração seguinte está **obviamente errada** dado que permite concluir que "todos os cavalos são brancos" (e, todos sabemos que há cavalos de outras cores).

> *Vamos mostrar que, qualquer que seja o número de cavalos que estejam numa cerca, se existir algum cavalo branco entre eles então todos os cavalos nessa cerca são brancos.*
>
> *Prova: Para isso vamos mostrar que as duas condições (i) e (ii) do princípio de indução se verificam.*

(i) Como deve estar pelo menos um cavalo branco na cerca, podemos afirmar que se só existir um cavalo na cerca é branco (pois terá de estar um branco).

(ii) Fixemos um $k \in \mathbb{N}$, e suponhamos que se existirem k cavalos numa cerca qualquer, estando pelo menos um cavalo branco entre eles, então os k cavalos são brancos. Consideramos agora uma cerca com $k+1$ cavalos, sendo branco pelo menos um deles. Retiremos um cavalo da cerca deixando ficar um branco. Dado que estão k cavalos na cerca, estando um branco entre eles, segue pela hipótese de indução, que os k cavalos na cerca são brancos. Resta mostrar que o cavalo que retirámos primeiramente também era branco. Retiremos um dos k cavalos que está na cerca (é indiferente qual se retira porque todos são brancos), e voltemos a colocar o que retirámos primeiramente. Mais uma vez, pela hipótese, concluimos que os k cavalos que estão agora na cerca são brancos.

Assim, pelo princípio de indução matemática segue a validade da proposição. (c.q.d)

Sabemos que "se existir algum cavalo branco numa cerca", também lá podem estar cavalos de outras cores. Logo, a afirmação que "mostrámos" é falsa. Mas, como o sistema dedutivo que estamos a seguir é consistente, não podemos deduzir proposições falsas, pelo que **há que encontrar um erro (vício) na prova dada**. Sabemos que se estiverem dois cavalos numa cerca e um deles for branco, tal não implica que o outro também seja branco. Por outras palavras, é falso $P(1) \Rightarrow P(2)$. Para localizarmos o erro, vamos então seguir em detalhe a prova de (ii) para $k = 1$.

Fixemos $k = 1$. Suponhamos (podemos sempre supor o que quisermos) que é sempre branco o cavalo que estiver sózinho numa cerca em que há pelo menos um cavalo branco. Consideramos agora uma cerca onde estão dois cavalos sendo branco pelo menos um deles. Retiremos um cavalo da cerca deixando ficar o branco. Dado que está um só cavalo na cerca, estando um branco na cerca, segue pela hipótese de indução, que o cavalo na cerca é branco. Resta mostrar que o cavalo que retirámos primeiramente também era branco. Retiremos o cavalo que está na cerca, e voltemos a colocar o que retirámos primeiramente. Agora não podemos aplicar a hipótese, pois só está na cerca o cavalo que acabámos de lá colocar, e não sabemos de que cor é (por isso não podemos afirmar que haja algum cavalo branco na cerca).

A prova de (ii) é válida para $k \geq 2$, ou seja é verdade que qualquer que seja $k \geq 2$, se forem brancos os k cavalos que estiverem numa qualquer cerca onde está pelo menos um branco, então são brancos os $k+1$ cavalos que estiverem numa qualquer cerca onde está pelo menos um branco. Conclusão: só não se pode mostrar que "todos os cavalos são brancos" porque $P(1) \not\Rightarrow P(2)$.

O exemplo seguinte ilustra alguns erros técnicos comuns, mas a evitar.

Exemplo 12 Vamos mostrar que $n^2 < 2^n$, para todo $n \geq 5$, por indução matemática. A demonstração seguinte está **tecnicamente incorreta** mas a proposição é verdadeira.

Mostremos que as condições (i) e (ii) do princípio de indução são satisfeitas.

(i) A condição verifica-se para $n = 5$, pois $n^2 = 25 < 32 = 2^n$.

(ii) Suponhamos que $n^2 < 2^n$, para todo $n \geq 5$. Para $n = n + 1$, temos $(n + 1)^2 = n^2 + 2n + 1 < n^2 + n^2 < 2^n + 2^n = 2^{n+1}$, porque $n^2 < 2^n$, por hipótese de indução, e porque $2n + 1 < n^2$, para $n \geq 5$, pois a função quadrática $x \to x^2 - 2x - 1$ é positiva se $x > 1 + \sqrt{2}$.

Portanto, por indução, concluimos que $n^2 < 2^n$, para todo $n \geq 5$.

A prova da condição (ii) contém erros técnicos graves:

- Começa por supor que "$n^2 < 2^n$, para todo $n \geq 5$". Mas, não o podemos fazer, pois é mesmo essa proposição que queremos mostrar! Teriamos sim de provar que $P(n) \Rightarrow P(n + 1)$, para todo $n \geq 5$, pelo que, como hipótese de indução, devemos assumir que para um certo $n \geq 5$, *fixo mas arbitrário*, se tem $P(n)$.

- Escrever "Para $n = n + 1 \ldots$" é falta de rigor. Como $n = n + 1$ não é satisfazível, a afirmação não faz sentido.

Para **corrigirmos a prova de (ii)**, devemos substituir a primeira linha por "(ii) Como hipótese de indução, suponhamos que, para $n \geq 5$ fixo, se tem $n^2 < 2^n$. Então, $(n + 1)^2 = n^2 + 2n + 1 < \ldots$".

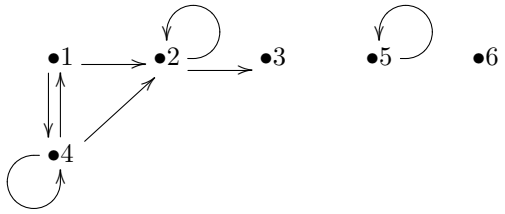
1.3 Noções elementares da teoria de grafos

Os grafos representam um modelo fundamental em computação, surgindo em problemas de caminhos (por exemplo, caminho mínimo ou máximo numa rede de transportes), representação de redes (por exemplo, de computadores, de tráfego rodoviário), descrição de sequência de programas, desenho de circuitos integrados, representação de relações (por exemplo, ordenação, emparelhamento), análise sintática de linguagens (árvores sintáticas), definição de diagramas de transição de máquinas (por exemplo, de autómatos finitos), etc.

1.3.1 Grafos dirigidos

Um **grafo dirigido** (digrafo) é definido por um par $G = (V, E)$, em que V é um conjunto de vértices (nós) e E um conjunto de ramos (arestas ou arcos) orientados que ligam pares de vértices de V, não existindo mais do que um ramo com a mesma orientação a ligar dois vértices. Quando o conjunto de vértices é **finito** (o que implica que o conjunto de ramos também o seja), o grafo é **finito**, e pode-se desenhar.

Exemplo 13 O grafo $G = (V, E)$, com $V = \{1, 2, 3, 4, 5, 6\}$ e $E = \{(1, 4), (1, 2), (2, 2), (2, 3), (4, 1), (4, 4), (5, 5)\}$, representa-se por:



Se $(x, y) \in E$, o vértice x é a **origem** e o vértice y o **fim** do ramo (x, y), dizendo-se que x e y são os **extremos** do ramo (x, y) e que o ramo é **incidente** em y. Os ramos (x, x), com origem e fim no mesmo vértice, chamam-se **lacetes**. No Exemplo 13, são três os ramos **incidentes no vértice** 2. Este vértice tem grau de entrada três e grau de saída dois. O **grau de entrada** de um vértice é o número de ramos que têm fim nesse vértice. O **grau de saída** é o número de ramos que têm origem nesse vértice. No mesmo exemplo, os vértices 5 e 6 são **vértices isolados**, pois não são origem nem fim de nenhum ramo com algum extremo noutro vértice.

Um **subgrafo** de um grafo $G = (V, E)$ é um grafo $G' = (V', E')$ em que $V' \subseteq V$ e $E' \subseteq E$. Também se pode definir subgrafo de um multigrafo (dirigido ou não). Alguns autores requerem que E' tenha todos os ramos de E que unem vértices em V', chamando *subgrafo parcial* se algum ramo que liga vértices em V' não pertencer a E'.

Percursos e Acessibilidade. Um **percurso (finito)** é uma sequência finita formada por um ou mais ramos do grafo, tal que o extremo final de qualquer ramo coincide com o extremo inicial do ramo seguinte na sequência. Um percurso num grafo dirigido pode ser identificado pela sequência de vértices por onde passa. No Exemplo 13, o percurso $(1, 4), (4, 1), (1, 2), (2, 2), (2, 2), (2, 3)$ pode ser dado pela sequência de vértices $(1, 4, 1, 2, 2, 2, 3)$.

A **origem do percurso** é a origem do primeiro ramo nesse percurso e o **fim do percurso** é o fim do último ramo no percurso. Um percurso com origem em v e fim em w é um **percurso de** v **para** w. Chama-se **circuito** a um percurso em que a origem e o fim coincidem. O **comprimento** (ou **ordem**) de um percurso (finito) é o número de ramos que o constituem. Assim, por exemplo, $(1, 4, 1, 2, 2, 2, 3)$ tem origem em 1, fim em 3 e comprimento seis, não sendo um circuito. Qualquer percurso de comprimento k envolve $k + 1$ vértices, não necessariamente distintos. No percurso $(1, 4, 1, 2, 2, 2, 3)$, alguns vértices **são visitados** mais do que uma vez, concretamente os vértices 1 e 2, e o ramo $(2, 2)$ é usado mais do que uma vez. Sendo u e v vértices de um grafo dirigido G, diz-se que v **é acessível de** u em G se e só se $u = v$ ou existe em G algum percurso de u para v.

A terminologia da teoria de grafos é pouco consistente, podendo ser necessário contextualizar cada termo na bibliografia em questão. Na terminologia moderna, um **caminho** é um percurso sem repetição de vértices e um **ciclo** é um percurso fechado que não tem repetição de vértices com excepção do inicial e final. Qualquer percurso com $|V|$ ramos é um ciclo ou contém um ciclo. Um grafo dirigido é **acíclico** se e só se não contém ciclos (nem lacetes).

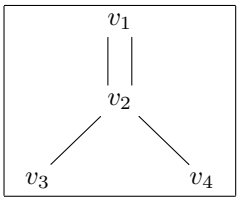
Multigrafos dirigidos. O conceito de grafo pode ser generalizado para permitir a existência de vários ramos com a mesma origem e fim. Um **multigrafo dirigido** é um terno (V, A, Φ), onde V é um conjunto de **vértices** (ou **nós**), A um conjunto de **ramos** (ou **arestas** ou **arcos**) e Φ é uma **função** de A em $V \times V$ que a cada ramo associa um par de vértices ($V \times V = \{(x, y) \mid x, y \in V\}$). Um percurso será definido por uma sequência de ramos, não podendo ser definido apenas pela sequência de vértices por onde passa.

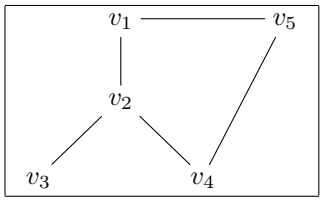
1.3.2 Grafos não dirigidos

Um **multigrafo não dirigido** é um terno (V, E, Φ) em que V é um conjunto de vértices (nós), E um conjunto de ramos (arestas) e Φ é uma função de E no conjunto de pares não ordenados de elementos de V. Escreve-se $\Phi(\alpha) = \langle u, v\rangle$. Usaremos $\langle u, v\rangle$ ou $\{u, v\}$ para representar um par não ordenado. Os nós u e v dizem-se **extremos** do ramo α. O **grau de um vértice** num multigrafo *não dirigido* é o número de ramos que têm esse vértice como extremo.

Um **grafo não dirigido** é um multigrafo não dirigido $G = (V, E, \Phi)$ em que Φ é uma função injetiva. Cada par de vértices não ordenado fica associado por Φ quando muito a um ramo. Por isso, é representado por $G = (V, E)$.

Exemplo 14 À esquerda está representado um multigrafo não dirigido que não é grafo dirigido, já que há mais do que um ramo a ligar v_1 e v_2. À direita temos um grafo não dirigido.

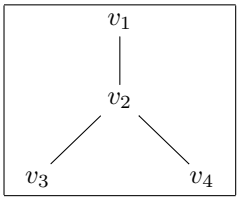


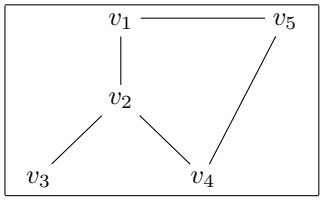


A um multigrafo não dirigido $G = (V, E, \Phi)$ podemos associar um multigrafo *dirigido* $G_A = (V, A, \Psi)$, substituindo cada ramo de E que una vértices diferentes por dois ramos orientados (com sentidos opostos). Se $\Phi(\alpha) = \langle u, v\rangle$ e $u \neq v$, existem dois ramos $\vec{\alpha_1}$ e $\vec{\alpha_2}$ em A tais que $\Psi(\vec{\alpha_1}) = (u, v)$ e $\Psi(\vec{\alpha_2}) = (v, u)$. O multigrafo G_A diz multigrafo adjunto de G. Se G for um grafo não dirigido então G_A é um grafo dirigido simétrico.

Um **percurso** (finito) num multigrafo não dirigido é uma sequência finita formada por um ou mais ramos do multigrafo que corresponde a um percurso no multigrafo adjunto. Num grafo não dirigido $(\langle v_0, v_1\rangle, \langle v_1, v_2\rangle, \ldots \langle v_{k-1}, v_k\rangle)$ representa um percurso **entre** v_0 e v_k. Se se indicar a sequência $(v_0, v_1, v_2, \ldots, v_{k-1}, v_k)$ de vértices por onde passa, pode ser entendido como um percurso de v_0 **para** v_k. O **comprimento** de um percurso (finito) é o seu número de ramos. Um **circuito** num grafo não dirigido é um percurso fechado $(\langle v_0, v_1\rangle, \langle v_1, v_2\rangle, \ldots \langle v_{k-1}, v_k\rangle)$ que v_0 e v_k coincidem. Um **ciclo** (por vezes designado *ciclo simples*) é um circuito que não tem repetição de vértices com excepção do inicial e final. Um grafo não dirigido é **acíclico** se e só se não contém ciclos (nem lacetes). Dados $u, v \in V$, diz-se que v **é acessível de** u em G se e só se $u = v$ ou existe em G algum percurso entre u e v.

Exemplo 15 O grafo representado abaixo, à esquerda tem circuitos mas é acíclico, pois não tem ciclos. O grafo representado à direita não é acíclico porque contém um ciclo, nomeadamente, $(v_1, v_2, v_4, v_5, v_1)$.



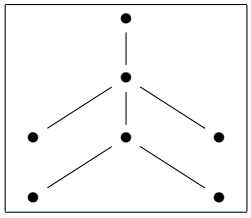


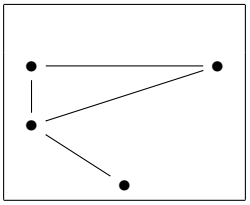
1.3.3 Caso particular: Árvores

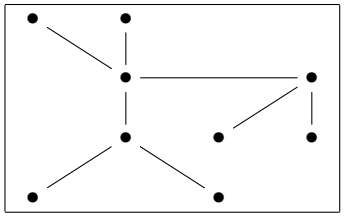
Um grafo não dirigido diz-se **conexo** se e só se qualquer vértice do grafo é acessível de qualquer outro vértice. Uma **árvore** é um grafo não dirigido $G = (V, E)$, conexo e finito, com $|V| - 1$ ramos. Uma árvore degenerada é uma árvore com apenas um vértice e, portanto, zero ramos. Numa árvore não degenerada, os vértices com grau 1 chamam-se **folhas** e os de grau superior chamam-se *nós internos* ou simplesmente **nós**. Na Proposição 5 recordamos algumas propriedades das árvores que podem ser usadas como definições alternativas.

Proposição 5 *As seis condições seguintes são equivalentes. Árvore não degenerada é um grafo não dirigido finito, com $n > 1$ vértices, e (i) conexo e acíclico; (ii) tal que entre cada par de vértices (distintos) existe um só caminho; (iii) conexo, e se se retirar algum ramo ($\langle u, v\rangle$, $u \neq v$) o grafo deixa de ser conexo; (iv) conexo e se se juntar mais algum ramo sem alterar o conjunto de vértices, o grafo fica com um e um só ciclo; (v) acíclico e com $n - 1$ ramos; e (vi) conexo e com $n - 1$ ramos.*

Exemplo 16 O grafo apresentado ao centro não é uma árvore e os restantes são.

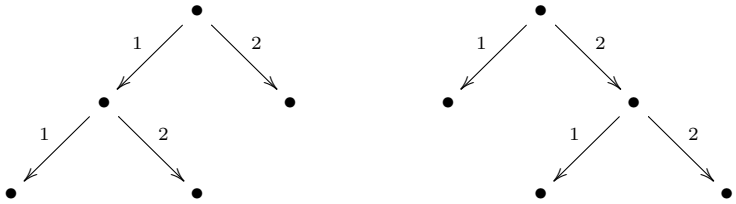






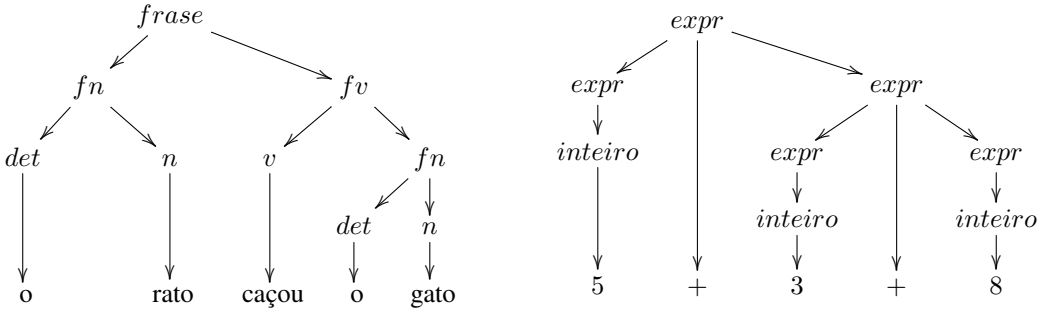
Uma **árvore com raíz** é um grafo dirigido com n vértices e $n - 1$ ramos e tal que existe um e um só vértice do qual todos são acessíveis. Tal vértice chama-se a **raíz** da árvore. As folhas numa árvore com raíz são os vértices com grau de saída zero. A raíz da árvore é o único vértice com grau de entrada zero. Numa árvore com raíz os **filhos** ou **descendentes *diretos* de um nó** v são os extremos finais de ramos com origem em v. Os **descendentes** de um nó são os filhos desse nó e os descendentes dos filhos desse nó. Um nó é **pai** dos seus filhos.

Uma **árvore ordenada** (ou **orientada**) é uma árvore com raíz e tal que o conjunto de ramos que saem de cada vértice está *totalmente ordenado*. As árvores seguintes poderiam iguais se se considerasse que eram árvores com raíz, mas são diferentes se se considerar que são árvores ordenadas (e $1 < 2$).



Os valores atribuídos aos ramos especificam a ordem subentendida. Em geral, não se colocam esses valores pois convenciona-se que a representação já obedece a essa ordem: *ou o ramo mais à esquerda é o maior ou é o menor.* Uma árvore ordenada diz-se **árvore n-ária** se o grau de saída de cada vértice não excede n. As árvores das figura anterior são árvores **binárias**. Numa árvore binária é usual falar na **sub-árvore direita** e na **sub-árvore esquerda** de um nó (tem por raíz o filho direito e esquerdo do nó).

Exemplo 17 As *árvores sintácticas* (ou de derivação) são árvores ordenadas. Voltaremos a considerá-las no Capítulo 5.



1.4 Relações binárias

Sejam A e B conjuntos. Designa-se por **produto cartesiano** de A por B, e denota-se por $A \times B$, o conjunto dos pares ordenados (a, b) tais que $a \in A$ e $b \in B$, ou seja

$$A \times B = \{(a, b) \mid a \in A \text{ e } b \in B\}$$

Qualquer subconjunto R do produto cartesiano de A por B diz-se **relação binária de A em B**. A notação $a\,R\,b$ será usada com o mesmo significado de $(a, b) \in R$.

Como as relações binárias são conjuntos (de pares ordenados), podemos definir a **união** e a **intersecção** de relações

binárias e ainda a **relação complementar**. Sendo $R, S \subseteq A \times B$:

$$\begin{aligned} R \cup S &= \{(a,b) \in A \times B \mid (a,b) \in R \ \vee \ (a,b) \in S\} \\ R \cap S &= \{(a,b) \in A \times B \mid (a,b) \in R \ \wedge \ (a,b) \in S\} \\ \overline{R} &= \{(a,b) \in A \times B \mid (a,b) \notin R\} = (A \times B) \backslash R \end{aligned}$$

Definimos também a noção de **inversa** e de **composta**. A relação **inversa** de R, representada por R^{-1}, é a relação de B em A definida pelo conjunto

$$R^{-1} = \{(b,a) \ \mid \ (a,b) \in R\}.$$

Para duas relações $R \subseteq A \times B$ e $S \subseteq B \times C$, a relação **composta** de R e S, representa-se por RS , e é uma relação binária de A em C definida por

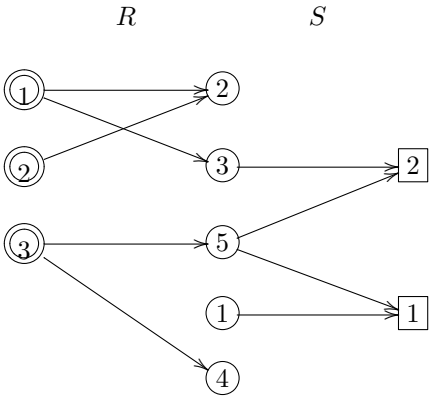
$$RS = \{(a,c) \ \mid \ \text{existe } b \in B \text{ tal que } (a,b) \in R \text{ e } (b,c) \in S\}.$$

Em alternativa podiamos escrever $S \circ R$, que se lê S após R, à semelhança da notação usual para *composição de funções*. Na secção 1.4.1, veremos que as funções correspondem a relações binárias que satisfazem condições adicionais. A *relação inversa* corresponde à noção de *imagem recíproca*, que associa a cada transformado os elementos do domínio que o têm por imagem.

Exemplo 18 Seja $A = \{1,2,3\}$, $B = \{1,2,3,4,5\}$, $C = \{1,2\}$, e sejam $R \subseteq A \times B$ e $S \subseteq B \times C$ dadas por

$$R = \{(1,2),(1,3),(2,2),(3,4),(3,5)\} \qquad S = \{(1,1),(3,2),(5,1),(5,2)\}$$

A relação R^{-1} é $\{(2,1),(3,1),(2,2),(4,3),(5,3)\}$. A composta, RS é dada por $RS = \{(1,2),(3,1),(3,2)\}$, como se conclui do esquema seguinte.



De facto, tem-se: $1\ RS\ 2$ pois $(1,3) \in R \wedge (3,2) \in S$; $(3,1) \in RS$ porque $(3,5) \in R \wedge (5,1) \in S$; $(3,2) \in RS$ pois $(3,5) \in R \wedge (5,2) \in S$; $(1,1) \notin RS$ pois $\forall y \in B \ \ (1,y) \notin R \vee (y,1) \notin S$ (de facto, $(1,y) \in R \Leftrightarrow y = 2 \vee y = 3$, mas $(2,1) \notin S \wedge (3,1) \notin S$); Do mesmo modo se verifica que, $(2,1) \notin RS$, $(2,2) \notin RS$.

Como $RS \subseteq \{1,2,3\} \times \{1,2\}$, fizemos uma análise exaustiva de todos os pares possíveis para determinar os que pertenciam a RS. Contudo, quando os conjuntos são finitos, as relações podem ser representadas por matrizes booleanas e a matriz da composta RS é dada pelo produto das matrizes que representam as relações R e S, usando a disjunção e a conjunção em vez da soma e produto usuais. Tal simplifica a sua determinação.

1.4.1 Noção de função e propriedades

Uma **função** de A em B é uma relação binária f de A em B, tal que, para todo $a \in A$, existe apenas um $b \in B$ tal que $a\ f\ b$, ou seja, $\forall a \in A\ \exists^1 b \in B\ \ a\ f\ b$.

Em vez de $a\ f\ b$, é usual escrever $f(a) = b$, dizendo-se que b é a **imagem** de a por f. O conjunto A é o **domínio** da função e B é o **conjunto de chegada**. O subconjunto de B formado pelas imagens dos elementos de A diz-se **imagem de A por f**, ou **contradomínio** de f. A imagem de A por f denota-se por $f(A)$. Uma função diz-se:

- **injetiva** sse não existem dois elementos de A (distintos) que tenham a mesma imagem.
- **sobrejetiva** sse qualquer elemento de B é imagem de algum elemento de A, ou seja $f(A) = B$.
- **bijetiva** sse é sobrejetiva e injetiva.

Assim, para uma função $f \subseteq A \times B$, a relação $f^{-1} \subseteq B \times A$ é uma função se e só se f é bijetiva. Uma função diz-se **invertível** se a sua inversa é uma função.

Uma **função parcial** de A em B é uma relação binária f de A em B tal que se um elemento $a \in A$ tem imagem por f então essa imagem é única. Notar que a diferença relativamente à definição de função é não se impor a condição de que todos os elementos de A tenham imagens por f.

1.4.2 Relações binárias definidas num conjunto

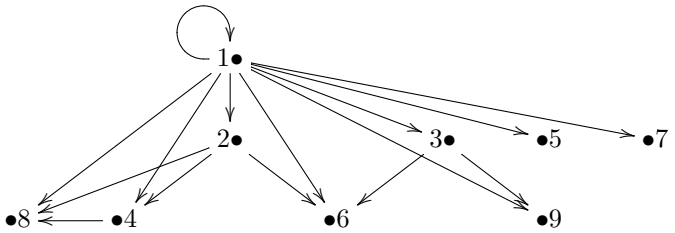
Uma **relação binária definida em** A (ou, simplesmente, **relação binária em** A) é qualquer relação binária R de A em A. Nesse caso, a relação binária R pode ser representada por um grafo dirigido $G = (A, R)$, que se chama **grafo da relação** R, cujos vértices são os elementos de A e os ramos são os pares ordenados que pertencem à relação R.

Qualquer relação R definida em A pode gozar ou não das propriedades seguintes.

- **reflexividade:** R é reflexiva sse $\forall x \in A\ \ xRx$, ou seja, não existe $x \in A$ tal que $(x,x) \notin R$.
- **simetria:** R é simétrica sse $\forall x, y \in A\ \ (xRy \Rightarrow yRx)$, ou seja, não existem $x, y \in A$ tais que $(x,y) \in R$ e $(y,x) \notin R$.
- **transitividade:** R é transitiva sse $\forall x, y, z \in A\ \ (xRy \wedge yRz) \Rightarrow xRz$, ou seja, não existem $x, y, z \in A$ tais que $(x,y) \in R$, $(y,z) \in R$ e $(x,z) \notin R$.

- **antissimetria:** R é antissimétrica sse $\forall x, y \in A\ ((xRy \wedge yRx) \Rightarrow x = y)$, ou seja, não existem $x, y \in A$, com $x \neq y$, tais que $(x, y) \in R$ e $(y, x) \in R$.

Exemplo 19 Seja $R = \{(x, y) \mid y \text{ é múltiplo de } x \text{ e } y \neq x\} \cup \{(1, 1)\}$, definida em $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Assim, $R = \{(1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7), (1,8), (2,4), (2,6), (2,8), (3,6), (3,9), (4,8)\}$ e pode ser representada esquematicamente pelo grafo:



Por análise do grafo, vemos que R não é reflexiva, não é simétrica, é antissimétrica e transitiva. Para a transitividade, basta ver que se existe um percurso de x para y de comprimento 2 então existe um arco de x para y.

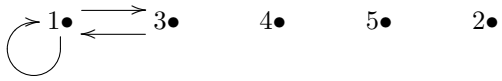
Exemplo 20 Seja $A = \{1, 2, 3, 4, 5\}$, e α, β, γ, e ϕ relações binárias em A, assim definidas:

$$\begin{aligned} \alpha &= \{(1,1), (2,2), (3,3), (4,4), (5,5)\} \\ \beta &= \{(1,1), (1,3), (3,1)\} \\ \gamma &= \{(1,1), (1,3)\} \\ \phi &= A \times A \end{aligned}$$

O grafo da relação α é:

A relação α é reflexiva, simétrica, transitiva, e antissimétrica. De facto, uma vez que $\alpha = \{(x, x) \mid x \in A\}$, é claro que $\forall x \in A\ \ (x, x) \in \alpha$, ou seja α é reflexiva (notar que o seu grafo tem lacetes em todos os vértices). Por definição de α tem-se $\forall x, y \in A\quad (\ (x, y) \in \alpha \Rightarrow x = y$. Donde se conclui que, $\forall x, y \in A\quad (\ (x, y) \in \alpha \Rightarrow (y, x) \in \alpha\ )$ ou seja, α é simétrica. Do mesmo modo, conclui-se que $\forall x, y, z \in A\quad (\ (x, y) \in \alpha \wedge (y, z) \in \alpha \Rightarrow (x, z) \in \alpha\ )$ ou seja α é transitiva. E ainda, $\forall x, y \in A\quad (\ (x, y) \in \alpha \wedge (y, x) \in \alpha \Rightarrow x = y\ )$ ou seja, α é antissimétrica.

O grafo da relação β é:



β não é reflexiva (porque por exemplo $5 \in A \wedge (5, 5) \notin \beta$), é simétrica (porque os únicos pares (x, y) que estão em β, com $x \neq y$, são $(1, 3)$ e $(3, 1)$), não antissimétrica (porque $1 \neq 3 \wedge (1, 3) \in \beta \wedge (3, 1) \in \beta$), e não transitiva (porque $(3, 1) \in \beta \wedge (1, 3) \in \beta \wedge (3, 3) \notin \beta$).

γ é transitiva, antissimétrica, não reflexiva, e não simétrica (Justificar!).

ϕ é reflexiva, simétrica, transitiva e não antissimétrica.

1.4.3 Relações de ordem parcial e total

Uma **relação de ordem parcial** definida num conjunto A é uma relação binária reflexiva, antissimétrica e transitiva.

Exemplo 21 A relação $\leq$, definida em $\mathbb{R}$ por $x \leq y$ se e só se existe $r \in \mathbb{R}_0^+ \;\; y = x + r$, é de ordem parcial. A relação "*divide*", denotada por $|$, definida em $\mathbb{Z}$ por $x|y$ se e só se existe $k \in \mathbb{Z}$ tal que $y = kx$, é de ordem parcial. Sendo A um conjunto, a relação de inclusão $\subseteq$ definida em 2^A é de ordem parcial. A sua inversa é $\supseteq$ e também é de ordem parcial.

É usual representar as relações de ordem parcial por $\preceq$. Um **conjunto parcialmente ordenado** é um par $(A, \preceq)$, em que A é um conjunto e $\preceq$ uma relação de ordem parcial em A. Um conjunto parcialmente ordenado $(A, \preceq)$ diz-se **conjunto totalmente ordenado** sse quaisquer que sejam x e y de A se tem $x \preceq y$ ou $y \preceq x$.

Exemplo 22 $(\mathbb{R}, \leq)$ é totalmente ordenado, já que $\forall x, y \in \mathbb{R}\ (x \leq y \vee y \leq x)$. Para a relação "divide", $(\mathbb{Z}, |)$ não é totalmente ordenado. Se A tiver pelo menos dois elementos, então $(2^A, \subseteq)$ não é totalmente ordenado.

1.4.4 Relações de equivalência

Uma **relação de equivalência** R em A é uma relação binária em A que é reflexiva, simétrica e transitiva. Se $(x, y) \in R$, então x e y dizem-se **equivalentes** (para R ou segundo R). A **classe de equivalência de** x em R é o conjunto dos elementos de A que são equivalentes a x segundo R. Se a denotarmos por C_x, então $C_x = \{y \in A \;\mid\; (x, y) \in R\}$. O **conjunto quociente de** A **por** R é o conjunto das classes de equivalência R e representa-se por A/R. Se o número de classes é finito, R diz-se de **índice finito**.

Exemplo 23 Seja $\equiv_5$ a relação definida no conjunto dos inteiros por $x \equiv_5 y$ se e só se $x - y$ é múltiplo (inteiro) de 5, para $x, y \in \mathbb{Z}$. Por definição, $x - y$ é múltiplo de 5 se existir $k \in \mathbb{Z}$ tal que $x - y = 5k$. A relação $\equiv_5$ é de equivalência:

- é reflexiva porque, qualquer que seja $x \in \mathbb{Z}$, tem-se $x - x = 0 = 0 \times 5$.
- é simétrica pois $x \equiv_5 y \;\Leftrightarrow (\exists k \in \mathbb{Z} \;\; x - y = 5k)$ e podemos tomar $k' = -k$ para concluir que existe $k' \in \mathbb{Z}$ tal que $y - x = 5k'$. Logo, $\forall x, y \in \mathbb{Z}\ (\ x \equiv_5 y \Rightarrow y \equiv_5 x)$.
- é transitiva porque que $\forall x, y, z \in \mathbb{Z}\ ((x \equiv_5 y \wedge y \equiv_5 z) \Rightarrow x \equiv_5 z\ )$. Se $x \equiv_5 y \wedge y \equiv_5 z$, existem $k, k' \in \mathbb{Z}$ tais que $x - y = 5k \wedge y - z = 5k'$. Assim, $x - z = (x - y) + (y - z) = 5(k + k')$. Logo, $x \equiv_5 z$.

São cinco as classes de equivalência de $\equiv_5$, sendo fixadas pelos cinco restos possíveis da divisão inteira por 5:

$C_0 = \{x \in \mathbb{Z} \mid 0 \equiv_5 x\} = \{x \mid \exists k \in \mathbb{Z} \;\; x = 5k\}$ $\quad$ $C_1 = \{x \in \mathbb{Z} \mid 1 \equiv_5 x\} = \{x \mid \exists k \in \mathbb{Z} \;\; x = 5k + 1\}$

$C_2 = \{x \in \mathbb{Z} \mid 2 \equiv_5 x\} = \{x \mid \exists k \in \mathbb{Z} \;\; x = 5k + 2\}$ $\quad$ $C_3 = \{x \in \mathbb{Z} \mid 3 \equiv_5 x\} = \{x \mid \exists k \in \mathbb{Z} \;\; x = 5k + 3\}$

$C_4 = \{x \in \mathbb{Z} \mid 4 \equiv_5 x\} = \{x \mid \exists k \in \mathbb{Z} \;\; x = 5k + 4\}$

O conjunto quociente é dado por $\mathbb{Z}/\equiv_5 \; = \; \{C_0, C_1, C_2, C_3, C_4\}$. A relação podia ser definida por: $x \equiv_5 y$ sse são iguais os restos da divisão inteira de x e y por 5. A notação mais habitual para esta relação é " $x \equiv y \pmod 5$ ".

É conhecido que, cada elemento de A pertence a uma e uma só classe de equivalência de R. Os elementos de C_x são equivalentes entre si e não são equivalentes a nenhum outro elemento de A, sendo $C_x = C_y$ sse $y \in C_x$. Qualquer elemento de uma classe de equivalência pode ser usado para identificar e representar a sua classe. Recordamos este resultado na Proposição 6.

Proposição 6 *(***Caraterização das classes de equivalência***) Seja R de equivalência em A, sendo $A \neq \emptyset$. Para cada $x \in A$, seja $C_x = \{y \in A \mid x\, R\, y\}$ a classe de equivalência de x. Então,*

(i) qualquer que seja $a \in A$, o conjunto C_a é não vazio.

(ii) se $C_a \cap C_b \neq \emptyset$ então $C_a = C_b$.

(iii) o conjunto A é a união dos conjuntos C_a, com $a \in A$.

Relações de equivalência e partições

Uma **partição de** A é um conjunto de subconjuntos de A que são não vazios, disjuntos dois a dois, e cuja união é A.

Exemplo 24 Para $A = \{1, 2, 3, 4\}$, o conjunto $P_1 = \{\{1, 2\}, \{3\}, \{4\}\}$ define uma partição de A, mas o conjunto $P_2 = \{\{1, 2\}, \{3, 2\}, \{4\}\}$ não, já que $\{1, 2\} \cap \{3, 2\} \neq \emptyset$. Também, $P_3 = \{\{1, 2\}, \{3\}\}$ não define partição de A já que $4 \in A$ e 4 não pertence a nenhum dos subconjuntos de A que constituem P_3.

O resultado que recordamos a seguir estabelece uma ligação intrínseca entre partições e relações de equivalência.

Proposição 7 *O conjunto quociente de uma relação de equivalência R definida em A constitui uma partição do conjunto A. Reciprocamente, para qualquer partição Π de A, existe uma e uma só relação de equivalência R em A, tal que $A/R = \Pi$. Tal relação é dada por $R = \{(x, y) \mid x$ e y pertencem a um mesmo elemento de $\Pi\}$.*

1.4.5 Fecho de uma relação binária para uma propriedade

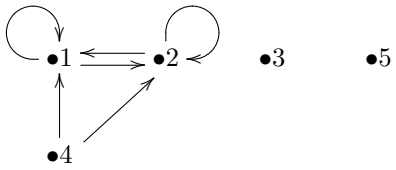
Seja R uma relação binária em A. Se existir alguma relação binária em A que contenha R e goze da propriedade P, então a *menor* das relações (no sentido da inclusão de conjuntos) que satisfazem tal condição chama-se **fecho da relação R para a propriedade P**. Se R_P denotar o fecho da relação R para a propriedade P então $R \subseteq R_P \subseteq A \times A$ e se S for uma qualquer relação em A que goze da propriedade P e contenha R, então S contém também R_P, isto é $R \subseteq R_P \subseteq S \subseteq A \times A$.

Exemplo 25 Seja $A = \{1,2,3,4,5\}$ e sejam $\theta = \{(1,1),(2,2),(3,3),(1,2)\}$ e $\beta = \{(1,1),(1,2),(2,1),(4,1)\}$ relações binárias em A, cujos grafos são:

O fecho reflexivo de θ é $\theta \cup \{(4,4),(5,5)\}$ e o de β é $\beta \cup \{(2,2),(3,3),(4,4),(5,5)\}$, o que corresponde a acrescentar lacetes em todos os vértices.

O fecho simétrico de θ é $\theta \cup \{(2,1)\}$ e o de β é $\beta \cup \{(1,4)\}$.

O fecho transitivo de θ é θ e o de β é $\beta \cup \{(2,2),(4,2)\}$:



Podemos verificar que, por exemplo, o fecho simétrico e transitivo de β é $\beta \cup \{(2,2),(4,2),(1,4),(2,4),(4,4)\}$.

Não faz sentido falar em fecho antissimétrico. Se uma relação não é antissimétrica, qualquer relação que a contém também não é antissimétrica. É fácil mostrar que o fecho reflexivo de R é dado por $R \cup \mathcal{I}_A$, onde $\mathcal{I}_A$ é a relação de identidade em A, isto é, $\mathcal{I}_A = \{(a,a) \mid a \in A\}$,e que o fecho simétrico de R é a relação $R \cup R^{-1}$. O fecho transitivo denota-se por R^+. Enunciamos a seguir algumas propriedades importantes do fecho transitivo.

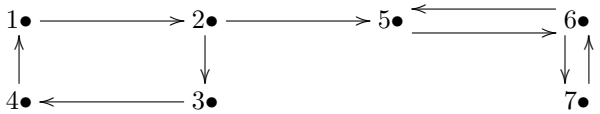
Fecho transitivo e percursos em grafos. Para $R \subseteq A \times A$, define-se a relação R^i, para cada $i \in \mathbb{Z}^+$, por:

$$\begin{aligned} R^1 &= R \\ R^i &= RR^{i-1}, \quad i \geq 2 \quad \text{(composta de } R \text{ e } R^{i-1}) \end{aligned}$$

Dizer que uR^iv, isto é, que $(u,v) \in R^i$, equivale a dizer que existe um percurso com i ramos de u para v no grafo de R. Mostra-se que xR^+y sse existir um percurso de x para y no grafo de R. Logo, a relação de acessibilidade no grafo de R é traduzida pelo fecho reflexivo e transitivo de R, o qual se pode denotar por $R^\star$. Apresentamos a seguir alguns resultados sobre R^+ e $R^\star$.

- R é transitiva se e só se $R^2 \subseteq R$;
- O fecho transitivo de R, que se denota por R^+, é dado por $\cup_{i\in\mathbb{Z}^+} R^i$, concluindo-se que uR^+v se e só se existir um percurso de u para v no grafo de R.
- Quando $|A| = n$ é finito, então $R^+ = \cup_{i=1}^{n} R^i$, pois, os ciclos com n ramos (se existirem) podem acrescentar informação relevante, mas não os percursos com mais de n ramos.
- Definindo $R^0 = I_A$, o fecho reflexivo e transitivo é dado por $R^\star = \cup_{i\in\mathbb{N}} R^i$. O vértice u é acessível do vértice v no grafo de R sse $(u, v) \in \mathcal{I}_A \cup R^+ = R^\star$.

Exemplo 26 Para a relação binária R em $A = \{1, 2, 3, 4, 5, 6, 7\}$, representada pelo grafo seguinte, tem-se:.



- $(1, 4) \in R^3$ pois $(1, 2) \in R \wedge (2, 3) \in R \wedge (3, 4) \in R$.
- $(1, 1) \in R^4$ pois $(1, 4) \in R^3 \wedge (4, 1) \in R$.
- $(3, 3) \in R^4$ pois $3\,R\,4 \ \wedge\ 4\,R\,1 \ \wedge\ 1\,R\,2 \ \wedge\ 2\,R\,3$.
- $(5, 5) \in R^+$ pois $(5, 5) \in R^2$ já que $5\,R\,6 \ \wedge\ 6\,R\,5$.
- Como $3\,R\,4 \ \wedge\ 4\,R\,1 \ \wedge\ 1\,R\,2 \ \wedge\ 2\,R\,5 \ \wedge 5\,R\,6 \ \wedge\ 6\,R\,7$, tem-se $(3, 7) \in R^6 \subseteq R^+$.
- $R^+ = (A \times A) \setminus (\{5, 6, 7\} \times \{1, 2, 3, 4\})$, não se conseguindo aceder dos vértices 5, 6 e 7 aos restantes.

Questões de notação. No próximo capítulo, introduzimos a notação $R^\star$ para denotar o fecho de Kleene de uma linguagem R. Simultaneamente, para formalizar alterações de estado numa máquina $\mathcal{A}$, usaremos $\vdash^\star_{\mathcal{A}}$ para denotar o fecho reflexivo e transitivo de uma relação binária $\vdash_{\mathcal{A}}$. Assim, como já referimos anteriormente, é importante identificar a interpretação que a notação tem em cada contexto.

Capítulo 2

Introdução às Linguagens Formais

2.1 Alfabeto, palavra e linguagem

Um **alfabeto** Σ é um qualquer conjunto finito e não vazio. Os elementos de um alfabeto dizem-se **símbolos**. Uma **palavra** (ou ainda, **frase** ou **sequência**) de alfabeto Σ é uma sequência finita de símbolos de Σ. As palavras podem ter zero ou mais símbolos. A **palavra vazia**, que representamos por ε, é a palavra com zero símbolos. O número de símbolos que constituem uma palavra α é o **comprimento** de α e denota-se por $|\alpha|$.

Uma **linguagem formal** de alfabeto Σ é um conjunto qualquer de palavras de alfabeto Σ. Se o conjunto for vazio, a linguagem diz-se **linguagem vazia**.

Exemplo 27 O conjunto das letras do abecedário $\{a, b, c, d, f, g, h, i, j, l, \ldots, v, x, z\}$ é um alfabeto. O conjunto das palavras que se podem encontrar num dicionário de Língua Portuguesa é uma linguagem de um alfabeto que inclui os símbolos anteriores e ainda, por exemplo, `-`, `é`, `ã`, ... Sabemos que `destruiu`, `o`, `gato`, `leu` e `livro` são palavras dessa linguagem. O conjunto das frases sintaticamente corretas (de acordo com as regras gramaticais de Português) é uma outra linguagem (sobre um alfabeto que inclui o carater espaço). Usando a terminologia anterior, dizemos que

```
o gato leu o livro
o gato destruiu o livro
o livro destruiu o gato
o livro leu o gato
```

são exemplos de *palavras* dessa linguagem, não o sendo as duas seguintes.

```
o o livro gato leu          livro o destruiu o leu gato
```

Por vezes, há que atender não só à boa formação sob o ponto de vista da **sintaxe**, mas também da **semântica**. Por exemplo, a frase "`o livro leu o gato`" está sintaticamente correta embora nos pareça semanticamente incorreta. Do mesmo modo, programas escritos numa linguagem de programação podem estar corretos sintaticamente mas incorretos semanticamente. Para que fique claro, focaremos principalmente aspetos da sintaxe de linguagens. Estudaremos ferramentas para caraterização de linguagens e classificaremos o poder computacional dessas ferramentas. Por isso, algumas das linguagens que serão introduzidas nos exemplos são abstratas.

Exemplo 28 As palavras de alfabeto $\{\mathtt{a}, \mathtt{b}\}$ que têm comprimento não superior a dois são ε, $\mathtt{a}$, $\mathtt{b}$, $\mathtt{aa}$, $\mathtt{ab}$, $\mathtt{ba}$, e $\mathtt{bb}$. Assim, $\{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}\}$ é uma linguagem de alfabeto $\{\mathtt{a}, \mathtt{b}\}$, a qual podemos ainda denotar, em compreensão, por $\{x \mid x \text{ é palavra de alfabeto } \{\mathtt{a}, \mathtt{b}\} \text{ e } |x| \leq 2\}$.

Exemplo 29 Seja L o conjunto das palavras de alfabeto $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ que são obtidas por aplicação das regras seguintes:

(r1) $\mathtt{aaa} \in L$

(r2) $\alpha\mathtt{bb}\beta \in L$, quaisquer que sejam $\alpha, \beta \in L$.

Por exemplo, `aaabbaaa` e `aaabbaaabbaaabbaaa` são palavras de L. Vamos demonstrar alguns resultados sobre as propriedades das palavras de L, usando indução matemática. Denotemos o número de `a`'s da palavra α por $n_{\mathtt{a}}(\alpha)$.

- Para todo p positivo e múltiplo de 3, existe $\gamma \in L$ tal que $n_{\mathtt{a}}(\gamma) = p$.

  Prova: Se p for 3, então podemos tomar γ como `aaa`. Seja agora $p > 3$ e múltiplo de 3. Se supusermos que para $p - 3$ (maior múltiplo de 3 que não excede p) existe $\gamma' \in L$ tal que $n_{\mathtt{a}}(\gamma') = p - 3$, podemos concluir que para p também existe $\gamma \in L$ tal que $n_{\mathtt{a}}(\gamma) = p$. De facto, basta tomar $\gamma = \mathtt{aaabb}\gamma'$, já que, por (r2), tem-se $\mathtt{aaabb}\gamma' \in L$. Assim, por indução matemática, deduzimos que, para todo $n \geq 1$, existe $\gamma \in L$ tal que $n_{\mathtt{a}}(\gamma) = 3n$. □

- Qualquer que seja $\alpha \in L$, tem-se $n_{\mathtt{a}}(\alpha)$ é positivo e múltiplo de 3.

  Prova: Qualquer que seja $\gamma \in L$, tem-se $\gamma = \mathtt{aaa}$ ou existem $\gamma_1, \gamma_2 \in L$ tais que $\gamma = \gamma_1\mathtt{bb}\gamma_2$. No primeiro caso, concluimos que $n_{\mathtt{a}}(\gamma)$ é múltiplo de 3. No segundo caso, como $n_{\mathtt{a}}(\gamma) = n_{\mathtt{a}}(\gamma_1) + n_{\mathtt{a}}(\gamma_2)$, podemos concluir que $n_{\mathtt{a}}(\gamma)$ é múltiplo de 3 se $n_{\mathtt{a}}(\gamma_1)$ e $n_{\mathtt{a}}(\gamma_2)$ forem múltiplos de 3, o que se terá se supusermos, como hipótese de indução (forte), que "$n_{\mathtt{a}}(\alpha)$ é múltiplo de 3, qualquer que seja $\alpha \in L$ com menor número de símbolos do que γ. " □

- $L = \{\mathtt{aaa}\} \cup \{\mathtt{aa}(\mathtt{bbaaa})^n \mid n \geq 1\}$, onde $\mathtt{aa}(\mathtt{bbaaa})^n$ denota $\mathtt{aa}\underbrace{\mathtt{bbaaa}\ldots\mathtt{bbaaa}}_{n \text{ vezes}}$.

  Deixamos, como exercício, a prova, por indução sobre n, de que

$$L \supseteq \{\mathtt{aaa}\} \cup \{\mathtt{aa}(\mathtt{bbaaa})^n \mid n \geq 1\}$$

e vamos justificar que

$$L \subseteq \{\texttt{aaa}\} \cup \{\texttt{aaa}(\texttt{bbaaa})^n \mid n \geq 1\}.$$

Se $\gamma \in L$ então $\gamma = \texttt{aaa}$ ou existem $\gamma_1, \gamma_2 \in L$ tais que $\gamma = \gamma_1 \texttt{bb} \gamma_2$. No segundo caso, γ tem pelo menos quatro símbolos. Como hipótese de indução, suponhamos que, para qualquer palavra $\alpha \in L$ com menos símbolos do que γ, existe $m \geq 0$ tal que α é da forma

$$\texttt{aaa}\underbrace{\texttt{bbaaa}\ldots\texttt{bbaaa}}_{m \text{ vezes}}$$

e sejam $m_1, m_2 \geq 0$ os valores m para γ_1 e γ_2, respetivamente. Vem,

$$\gamma = \texttt{aaa}\underbrace{\texttt{bbaaa}\ldots\texttt{bbaaa}}_{m_1 \text{ vezes}}\texttt{bbaaa}\underbrace{\texttt{bbaaa}\ldots\texttt{bbaaa}}_{m_2 \text{ vezes}}$$

ou seja, $\gamma = \texttt{aaa}(\texttt{bbaaa})^{m_1+m_2+1}$

Neste exemplo, seguimos uma prova por indução sobre o comprimento das palavras. Em particular, por fim, mostrámos que, para todo $n \geq 3$ e todo $\gamma \in L$ com comprimento n, existe $m \in \mathbb{N}$ tal que $\gamma = \texttt{aaa}(\texttt{bbaaa})^m$. Ou seja,

$$\forall n \geq 3 \ \ \forall \gamma \in L \ \ (|\gamma| = n \Rightarrow \exists m \in \mathbb{N} \ \ \gamma = \texttt{aaa}(\texttt{bbaaa})^m).$$

Exemplo 30 Seja L a linguagem de alfabeto $\Sigma = \{\texttt{p}, \wedge, \vee, \neg, \Rightarrow, \texttt{(}, \texttt{)}\}$ assim definida indutivamente. As palavras de L são que se podem obter usando as regras (i) e (ii) um número finito de vezes:

(i) $\texttt{p} \in L$

(ii) Quaisquer que sejam $\alpha, \beta \in L$, tem-se $\texttt{(}\alpha\wedge\beta\texttt{)} \in L$, $\texttt{(}\alpha\vee\beta\texttt{)} \in L$, $\texttt{(}\alpha\Rightarrow\beta\texttt{)} \in L$ e $\texttt{(}\neg\alpha\texttt{)} \in L$.

As sequências seguintes são exemplos de palavras de L (que podemos entender como parte do cálculo proposicional):

$$\texttt{(p}\wedge\texttt{p)}$$

$$\texttt{((p}\wedge\texttt{p)}\vee\texttt{p)}$$

$$\texttt{(((p}\wedge\texttt{p)}\vee\texttt{p)}\Rightarrow\texttt{(}\neg\texttt{p))}$$

$$\texttt{(((p}\wedge\texttt{p)}\vee\texttt{p)}\Rightarrow\texttt{(}\neg\texttt{p))}\Rightarrow\texttt{((}\neg\texttt{p)}\vee\texttt{p))}$$

2.2 Operações com linguagens

Como as linguagens são conjuntos, podemos definir $L \cup M$ (**união**), $L \cap M$ (**interseção**), $L \setminus M$ (**diferença**), bem como $\overline{L} = \Sigma^\star \setminus L$ (**linguagem complementar de** L), para linguagens L e M sobre Σ.

$$L \cup M \quad = \quad \{x \mid x \in L \text{ ou } x \in M\}$$

$$\begin{aligned} L \cap M &= \{x \mid x \in L \text{ e } x \in M\} \\ L \setminus M &= \{x \mid x \in L \text{ e } x \notin M\} \\ \overline{L} &= \{x \mid x \in \Sigma^\star \text{ e } x \notin L\} \end{aligned}$$

Introduzimos a seguir duas novas operações: a concatenação e o fecho de Kleene.

2.2.1 Concatenação de palavras e concatenação de linguagens

Seja $\Sigma^\star$ o conjunto de todas as palavras de alfabeto Σ. Se $\alpha \in \Sigma^\star$ e $\beta \in \Sigma^\star$ então a palavra $\alpha\beta$, obtida por justaposição de β no fim de α, é a **concatenação** de α com β. A concatenação é uma operação binária em $\Sigma^\star$ (isto é, a concatenação de duas quaisquer palavras de $\Sigma^\star$ é uma palavra de $\Sigma^\star$). A concatenação é associativa, i.e., $\alpha(\beta\gamma) = (\alpha\beta)\gamma$, quaisquer que sejam $\alpha, \beta, \gamma \in \Sigma^\star$, o que dá significado à notação $\alpha\beta\gamma$. A concatenação tem elemento neutro. A palavra vazia é o seu elemento neutro, pois $\varepsilon\alpha = \alpha = \alpha\varepsilon$, qualquer que seja $\alpha \in \Sigma^\star$. Não introduzimos qualquer símbolo para representar o operador de concatenação.

O exemplo seguinte mostra que a concatenação de palavras não é comutativa.

Exemplo 31 A concatenação da palavra `flor` com a palavra `bela` é a palavra `florbela`. A concatenação da palavra `bela` com a palavra `flor` é `belaflor`.

Se $x \in \Sigma^\star$ é uma palavra, e $x = uvw$ para $u, v, w \in \Sigma^\star$, então u diz-se **prefixo** de x, w diz-se **sufixo**, e u, v, e w dizem-se **subpalavras** de x. Por exemplo, a palavra `abaa`, de alfabeto $\{\mathtt{a}, \mathtt{b}\}$, tem como subpalavras ε, `a`, `b`, `ab`, `ba`, `aa`, `aba`, `baa`, e `abaa`. Qualquer palavra é subpalavra, prefixo e sufixo de si mesma.

Exemplo 32 A linguagem L das sequências de parentesis curvos que são *bem formadas* é a linguagem de alfabeto $\Sigma = \{\,(\,,\,)\,\}$ constituída pelas palavras que se podem obter usando as regras (i), (ii) e (iii), quantas vezes se quiser.

(i) $() \in L$

(ii) Quaisquer que sejam $\alpha, \beta \in L$, tem-se $\alpha\beta \in L$

(iii) Para todo $\alpha \in L$, tem-se $(\alpha) \in L$

Podemos mostrar que L é o conjunto das sequências de parentesis que têm algum parentesis e tais que, em qualquer prefixo de uma tal sequência, o número de parentesis fechados não é maior do que o número de parentesis abertos.

Sendo L e M linguagens de alfabeto Σ, a **concatenação de L com M** é a linguagem de alfabeto Σ definida por:

$$LM = \{\alpha\beta \mid \alpha \in L \text{ e } \beta \in M\}.$$

Para $n \geq 1$, a linguagem das palavras obtidas por justaposição de n palavras de L é

$$L^n = \{w_1 \dots w_n \mid w_1 \in L \wedge \dots \wedge w_n \in L\}$$

sendo $L^1 = L$. Define-se também L^0 por $L^0 = \{\varepsilon\}$. Para $n \geq 0$, a linguagem L^n é definida pela recorrência $L^0 = \{\varepsilon\}$ e $L^n = LL^{n-1} = L^{n-1}L$, para $n \geq 1$.

Como vimos, sendo x uma palavra qualquer e $n \in \mathbb{N}$, usamos a notação x^n para representar $\underbrace{x \dots x}_{n \text{ vezes}}$.
Assim, $x^0 = \varepsilon$ e $x^n = x^{n-1}x = xx^{n-1}$, para $n \geq 1$.

Exemplo 33 *Seja* $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ *então:*

$$\begin{aligned}
\Sigma^2 &= \{\mathtt{aa}, \mathtt{ab}, \mathtt{bb}, \mathtt{ba}\} \\
\Sigma^3 &= \Sigma\Sigma^2 = \{wv \mid w \in \Sigma \wedge v \in \Sigma^2\} = \{\mathtt{aaa}, \mathtt{aab}, \mathtt{abb}, \mathtt{aba}, \mathtt{baa}, \mathtt{bab}, \mathtt{bbb}, \mathtt{bba}\} \\
\Sigma^n &= \{w \in \Sigma^\star \mid \ |w| = n\}, \ \textit{para } n \textit{ qualquer.}
\end{aligned}$$

Um parentesis sobre (subtilezas de) notação... Não é indiferente escrever "$\Sigma^n = \{w \in \Sigma^\star \mid \ |w| = n\}$, para n qualquer" (ou, equivalentemente, "$\{w \in \Sigma^\star \mid \ |w| = n\}$, com $n \in \mathbb{N}$") e escrever "$\{w \in \Sigma^\star \mid \ |w| = n \text{ e } n \in \mathbb{N}\}$". A linguagem $\{w \in \Sigma^\star \mid \ |w| = n\}$ é constituída pelas palavras que têm um dado comprimento n (para n fixo) e a linguagem $\{w \in \Sigma^\star \mid \ |w| = n \text{ e } n \in \mathbb{N}\}$ é $\Sigma^\star$.

Exemplo 34 *Seja* $L = \{\mathtt{a}, \mathtt{ab}, \mathtt{bb}\}$ *então:*

$$\begin{aligned}
L^2 &= \{\mathtt{aa}, \mathtt{aab}, \mathtt{abb}, \mathtt{aba}, \mathtt{abab}, \mathtt{abbb}, \mathtt{bba}, \mathtt{bbab}, \mathtt{bbbb}\} \\
L^3 &= LL^2 = \{wv \mid w \in L \wedge v \in L^2\} = \{\mathtt{aaa}, \mathtt{aaab}, \mathtt{aabb}, \mathtt{aaba}, \mathtt{aabab}, \mathtt{aabbb}, \mathtt{abba}, \mathtt{abbab}, \\
&\qquad \mathtt{abbbb}, \mathtt{abaa}, \mathtt{abaab}, \mathtt{ababb}, \mathtt{ababa}, \mathtt{ababab}, \mathtt{ababbb}, \mathtt{abbba}, \mathtt{abbbab}, \mathtt{abbbbb}, \\
&\qquad \mathtt{bbaa}, \mathtt{bbaab}, \mathtt{bbabb}, \mathtt{bbaba}, \mathtt{bbabab}, \mathtt{bbabbb}, \mathtt{bbbba}, \mathtt{bbbbab}, \mathtt{bbbbbb}\}
\end{aligned}$$

2.2.2 Fecho de Kleene de uma linguagem

O **fecho de Kleene** de uma linguagem L é o conjunto das palavras que se podem formar por justaposição de zero ou mais palavras de L. Representa-se por $L^\star$ e é definido por

$$L^\star = \{\varepsilon\} \cup \{w_1 \dots w_n \mid w_i \in L, \ 1 \leq i \leq n, \ n \in \mathbb{N}\} = \bigcup_{n \in \mathbb{N}} L^n.$$

Observemos que $\Sigma^\star$ não é mais do que o fecho de Kleene de Σ, sendo o conjunto de todas as palavras de alfabeto Σ.

Exemplo 35 O fecho de Kleene de algumas linguagens sobre $\Sigma = \{0, 1\}$:

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

A linguagem $\{1\}^\star$ é constituída pelas palavras que não têm 0's. A linguagem $\{01\}^\star$ é constituída pelas palavras de comprimento par que não têm 0's consecutivos nem 1's consecutivos e não começam por 1. A linguagem $\{000\}^\star$ é constituída pelas palavras que não têm 1's e cujo comprimento é múltiplo de três.

Exercício 2.2.1 Para ver que $\{000, 00000\}^\star = \{\varepsilon, 000, 00000, 000000\} \cup \{0^n \mid n \geq 8\}$ e, em particular, que qualquer inteiro maior ou igual a 8 é combinação de 3 e 5, suponhamos que se pretendia selar uma carta e só havia disponíveis selos de cinco e de três u.m. (unidades monetárias). Se não se limitar o número de selos que se pode usar, como é que se obtém cada uma das quantias 8, 9, 10, 11, 12, 13, 14, 15, 16, ...? Por exemplo, como $q = 8 = 5 + 3$, se trocarmos o selo de 5 por dois de 3, obtemos $9 = 3 + 3 + 3$. Dada uma solução para a quantia q, que regra aplicar para obter a quantia $q + 1$, por troca do menor número de selos dessa solução, para $q \geq 8$?

Lema 1 *Sejam L_1 e L_2 linguagens de alfabeto Σ. Se $L_1 \subseteq L_2$, então qualquer que seja $n \in \mathbb{N}$, $L_1^n \subseteq L_2^n$, e $L_1^\star \subseteq L_2^\star$.*

Prova: Queremos provar que se $L_1 \subseteq L_2$ então $L_1^n \subseteq L_2^n$, qualquer que seja $n \in \mathbb{N}$. Mostrar que $L_1^n \subseteq L_2^n$ equivale a mostrar que qualquer que seja x, se $x \in L_1^n$ então $x \in L_2^n$. Sabemos que L_1^n é, por definição, o conjunto das palavras obtidas por justaposição de n palavras de L_1. Assim, se $x \in L_1^n$ então x é justaposição de n palavras de L_1. Mas, como $L_1 \subseteq L_2$, qualquer palavra de L_1 é palavra de L_2, pelo que podemos concluir que se x é justaposição de n palavras de L_1, então x é justaposição de n palavras de L_2. Logo, se $x \in L_1^n$ então $x \in L_2^n$.

De modo análogo, podemos concluir que se x for uma justaposição de zero ou mais palavras de L_1 então x é justaposição de zero ou mais palavras de L_2. Ou seja, concluimos que, qualquer que seja x, se $x \in L_1^\star$ então $x \in L_2^\star$. Logo, se $L_1 \subseteq L_2$ então $L_1^\star \subseteq L_2^\star$. □

Lema 2 *Sejam L_1 e L_2 linguagens de alfabeto Σ. Se $\varepsilon \in L_1$ então $L_2 \subseteq L_1 L_2$, e se $\varepsilon \in L_2$ então $L_1 \subseteq L_1 L_2$.*

Prova: Por definição de $L_1 L_2$, tem-se $\varepsilon x \in L_1 L_2$, para todo $x \in L_2$, dado que $\varepsilon \in L_1$. Mas, como ε é o elemento neutro da concatenação, $\varepsilon x = x$. Logo, se $\varepsilon \in L_1$ então $x \in L_1 L_2$, qualquer que seja $x \in L_2$, pelo que $L_2 \subseteq L_1 L_2$. Do modo análogo, podemos mostrar que se $\varepsilon \in L_2$ então $L_1 \subseteq L_1 L_2$. □

Proposição 8 *Quaisquer que sejam as linguagens R, S e T de alfabeto Σ tem-se:*

(1) $R \cup S = S \cup R$ (2) $(R \cup S) \cup T = R \cup (S \cup T)$

(3) $R \cap S = S \cap R$ (4) $(R \cap S) \cap T = R \cap (S \cap T)$

(5) $R(ST) = (RS)T$ (6) $R(S \cup T) = RS \cup RT$

(7) $(R \cup S)T = RT \cup ST$ (8) $\emptyset^\star = \{\varepsilon\}$

(9) $(R^\star)^\star = R^\star$ (10) $(\{\varepsilon\} \cup R)^\star = R^\star$

(11) $(R^\star S^\star)^\star = (R \cup S)^\star$ (12) $\{\varepsilon\}R = R = R\{\varepsilon\}$

(13) $\emptyset R = \emptyset = R\emptyset$

Prova: Provamos apenas (9), (11), (12) e (13) ficando as restantes ao cuidado do leitor.

(9) $(R^\star)^\star = R^\star$.

Por definição de fecho de Kleene, tem-se $R^\star = (R^\star)^1 \subseteq \bigcup_{n\geq 0}(R^\star)^n = (R^\star)^\star$. Logo, $R^\star \subseteq (R^\star)^\star$.

Resta mostrar que também $(R^\star)^\star \subseteq R^\star$ ou seja, que $\forall w \ \ (w \in (R^\star)^\star \Rightarrow w \in R^\star)$. Ora, se $w \in (R^\star)^\star$ então w é justaposição de zero ou mais palavras de $R^\star$. Se $w = \varepsilon$, então $w \in R^\star$. Se $w \neq \varepsilon$ então, sem perda de generalidade (uma vez que ε é elemento neutro para a concatenação), podemos dizer que w foi obtido por justaposição de um certo número de palavras de $R^\star \setminus \{\varepsilon\}$. Por sua vez, por definição de $R^\star$, cada uma dessas palavras é justaposição de um certo número de palavras de R. Logo, se w é justaposição de palavras de $R^\star \setminus \{\varepsilon\}$ então w é justaposição de palavras de $R \setminus \{\varepsilon\}$, o que mostra que $(R^\star)^\star \subseteq R^\star$. □

(11) $(R^\star S^\star)^\star = (R \cup S)^\star$. Comecemos por mostrar que $(R^\star S^\star)^\star \subseteq (R \cup S)^\star$. Para isso, vamos justificar que $R^\star S^\star \subseteq (R \cup S)^\star$, o que é trivial. De facto, qualquer palavra de $R^\star S^\star$ é justaposição de uma sequência de zero ou mais palavras de R com uma sequência de zero ou mais palavras de S. Logo, qualquer palavra de $R^\star S^\star$ é uma sequência de zero ou mais palavras de $R \cup S$. Para concluir que $(R^\star S^\star)^\star \subseteq (R \cup S)^\star$, basta notar que se $R^\star S^\star \subseteq (R \cup S)^\star$ então, pelo Lema 1, vem $(R^\star S^\star)^\star \subseteq ((R \cup S)^\star)^\star$, e como $((R \cup S)^\star)^\star = (R \cup S)^\star$ (por (9)), conclui-se que $(R^\star S^\star)^\star \subseteq (R \cup S)^\star$.

Falta agora mostrar que $(R^\star S^\star)^\star \supseteq (R \cup S)^\star$, o que se pode concluir usando o Lema 1, se mostrarmos que $R^\star S^\star \supseteq R \cup S$. Para isso, note-se que como $\varepsilon \in R^\star$ e $\varepsilon \in S^\star$, se deduz, pelo Lema 2, que $S^\star \subseteq R^\star S^\star$ e $R^\star \subseteq R^\star S^\star$. Por outro lado, como $R \subseteq R^\star$ e $S \subseteq S^\star$, obtem-se $R \subseteq R^\star S^\star$ e $S \subseteq R^\star S^\star$. Donde, por definição de união de conjuntos, $R \cup S \subseteq R^\star S^\star$ □

(12) $\{\varepsilon\}R = R = R\{\varepsilon\}$. Por definição de concatenação, $\{\varepsilon\}R = \{\varepsilon y \mid y \in R\} = \{y \mid y \in R\} = R$, pois $\varepsilon y = y$, para todo y. Analogamente, podemos ver que $R\{\varepsilon\} = \{y\varepsilon \mid y \in R\} = R$. □

(13) $\emptyset R = \emptyset = R\emptyset$. Por definição de concatenação de lingugagens, $\emptyset R = \{xy \mid x \in \{\,\} \text{ e } y \in R\}$. Mas, nenhuma palavra xy se pode formar pois $\{\,\}$ não tem elementos e, portanto, $x \notin \{\,\}$, para todo x. Logo, $\emptyset R = \emptyset$. Analogamente, podemos concluir que $R\emptyset = \emptyset$. □

Capítulo 3

Linguagens Regulares e Autómatos Finitos

Por vezes, há vantagem em descrever o padrão das palavras de uma linguagem por uma abreviatura, que é uma expressão. Vamos introduzir um conjunto de expressões, designadas por **expressões regulares sobre** Σ, que permitem descrever algumas linguagens de alfabeto Σ. Iremos depois introduzir um modelo de máquinas para caraterização deste tipo de linguagens, designadas por **autómatos finitos**. No Capítulo 4, veremos que há linguagens que não podem ser descritas por expressões regulares ou, equivalentemente, que não podem ser reconhecidas por autómatos finitos.

3.1 Expressões regulares sobre um alfabeto

O conjunto das **expressões regulares sobre** Σ e o conjunto das linguagens que descrevem são assim definidos indutivamente:

(i) ε é uma expressão regular sobre Σ e descreve a linguagem $\{\varepsilon\}$;

(ii) $\emptyset$ é uma expressão regular sobre Σ e descreve a linguagem $\emptyset$;

(iii) Se $\mathtt{a} \in \Sigma$ então $\mathtt{a}$ é uma expressão regular sobre Σ e descreve a linguagem $\{\mathtt{a}\}$;

(iv) Se r e s são expressões regulares sobre Σ que descrevem as linguagens R e S, então $(r + s)$, (rs) e $(r^\star)$ são expressões regulares sobre Σ e descrevem $R \cup S$, RS e $R^\star$, respetivamente.

(v) As expressões regulares sobre Σ e as linguagens por elas descritas são todas e apenas as obtidas por (i)-(iv).

Definição 1 *Uma linguagem diz-se* **linguagem regular** *se e só se é descrita por uma expressão regular.*

Usaremos a notação $\mathcal{L}(r)$ para referir a linguagem descrita pela expressão regular r.

Ainda sobre (subtilezas de) notação... Para traduzir simbolicamente que *a palavra* 0000 *não pertence à linguagem descrita pela expressão regular* $(000)^\star(\varepsilon + 0^\star 1)$, não escrevemos $0000 \notin (000)^\star(\varepsilon + 0^\star 1)$ mas sim

$$0000 \notin \mathcal{L}((000)^\star(\varepsilon + 0^\star 1))$$

Exemplo 36 A tabela abaixo apresenta expressões regulares sobre $\Sigma = \{0, 1\}$ e as linguagens que descrevem.

expressão regular	linguagem descrita
$(0 + 1)$	$\{0, 1\}$
$(0^\star)$	$\{\varepsilon, 0, 00, 000, 0000, 00000, \ldots\} = \{$palavras sem 1' s$\}$
$((0^\star)(11))$	$\{11, 011, 0011, 00011, 000011, 0000011, \ldots\}$
$((01)^\star)$	$\{\varepsilon, 01, 0101, 010101, 01010101, 0101010101, \ldots\}$
$((0 + 1)^\star)$	$\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots\}$
$((1^\star)(0((0 + 1)^\star)))$	$\{0, 10, 01, 00, 000, 100, 010, 001, 110, 101, 011, \ldots\} = \{x \mid x$ tem algum $0\}$

Duas **expressões regulares** sobre Σ são **equivalentes** se e só se descrevem a mesma linguagem. Tal corresponde a definir uma relação de equivalência $\equiv$ no conjunto das expressões regulares sobre Σ por:

$$r \equiv s \quad sse \quad \text{as linguagens descritas por } r \text{ e } s \text{ são iguais, isto é, } r \equiv s \text{ sse } \mathcal{L}(r) = \mathcal{L}(s).$$

A proposição seguinte enuncia algumas propriedades que podem ser usadas para simplificar expressões regulares.

Proposição 9 *Quaisquer que sejam as expressões regulares r, s e t de alfabeto Σ tem-se:*

(1) $(r + s) \equiv (s + r)$
(2) $((r + s) + t) \equiv (r + (s + t))$
(3) $(r(st)) \equiv ((rs)t)$
(4) $(r(s + t)) \equiv ((rs) + (rt))$
(5) $((r + s)t) \equiv ((rt) + (st))$
(6) $(\emptyset^\star) \equiv \varepsilon$
(7) $((r^\star)^\star) \equiv (r^\star)$
(8) $((\varepsilon + r)^\star) \equiv (r^\star)$
(9) $(((r^\star)(s^\star))^\star) \equiv ((r + s)^\star)$
(10) $(\varepsilon r) \equiv r \equiv (r\varepsilon)$
(11) $(\emptyset r) \equiv \emptyset \equiv (r\emptyset)$

Prova: Segue imediatamente como corolário da Proposição 8, usando a definição de expressão regular e de linguagem por ela descrita, e denotando $\mathcal{L}(r)$, $\mathcal{L}(s)$ e $\mathcal{L}(t)$ por R, S e T. □

Usando uma convenção idêntica à aplicada a expressões aritméticas envolvendo soma, produto e potenciação, podemos abreviar as expressões retirando parentesis "desnecessários", o que facilita a sua interpretação. As regras de precedência para as operações $+$, "concatenação" e * correspondem às usadas nas expressões aritméticas para a soma, o produto e a potenciação, respetivamente.

A vantagem da introdução desta simplificação é claramente ilustrada pelos exemplos dados na Figura 3.1.

$(r + s)$	$r + s$
$((r + s) + t)$	$r + s + t$
$(r(st))$	rst
$(r(s + t))$	$r(s + t)$
$((rs) + (rt))$	$rs + rt$
$(\emptyset^\star)$	$\emptyset^\star$
$((r^\star)^\star)$	$(r^\star)^\star$
$(((r^\star)(s^\star))^\star)$	$(r^\star s^\star)^\star$
$(\mathtt{0} + \mathtt{1})$	$\mathtt{0} + \mathtt{1}$
$((\mathtt{0}^\star)(\mathtt{11}))$	$\mathtt{0}^\star\mathtt{11}$
$(((\mathtt{0}^\star)(\mathtt{11})) + ((\mathtt{10})\mathtt{1}))$	$\mathtt{0}^\star\mathtt{11} + \mathtt{101}$
$((\mathtt{01})^\star)$	$(\mathtt{01})^\star$
$((\mathtt{1}^\star)(\mathtt{0}((\mathtt{0} + \mathtt{1})^\star)))$	$\mathtt{1}^\star\mathtt{0}(\mathtt{0} + \mathtt{1})^\star$
$(((\mathtt{0} + \mathtt{1})^\star)((\mathtt{00})((\mathtt{0} + \mathtt{1})^\star)))$	$(\mathtt{0} + \mathtt{1})^\star\mathtt{00}(\mathtt{0} + \mathtt{1})^\star$
$((((\mathtt{0} + \mathtt{1})^\star)((\mathtt{00})\mathtt{0})) + (((\mathtt{01})^\star)(\mathtt{01})))$	$(\mathtt{0} + \mathtt{1})^\star\mathtt{000} + (\mathtt{01})^\star\mathtt{01}$

Figura 3.1: Expressão regular e abreviatura respetiva

Exemplo 37 Mais alguns exemplos de linguagens de alfabeto $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ e de expressões regulares que as descrevem.

linguagem	expressão regular
$\{\mathtt{0}, \mathtt{00}, \mathtt{000}\}$	$\mathtt{0} + \mathtt{00} + \mathtt{000}$
$\{\mathtt{000}, \mathtt{00000}\}^\star$	$(\mathtt{000} + \mathtt{00000})^\star$
$\{\mathtt{0}^{3n+1} \mid n \in \mathbb{N}\}$	$\mathtt{0}(\mathtt{000})^\star$
$\{w \in \Sigma^\star \mid w$ não tem 1's e tem número par de 0's$\}$	$(\mathtt{00})^\star$
$\{w \in \Sigma^\star \mid w$ tem pelo menos um 1 depois de cada 0$\}$	$(\mathtt{01} + \mathtt{1})^\star$
$\{w \in \Sigma^\star \mid w$ tem pelo menos dois 1's entre cada par de 0's$\}$	$(\mathtt{1} + \mathtt{011})^\star(\varepsilon + \mathtt{0} + \mathtt{01})$

A linguagem das palavras que têm pelo menos dois 1's entre cada par de 0's pode ser descrita pela expressão regular $(1 + 011)^\star(\varepsilon + 0 + 01)$, porque se $w \in \{0,1\}^\star$ tiver dois ou mais 0's, então, para que entre cada duas ocorrências de 0 haja pelo menos dois 1's, todos os 0's de w vêm seguidos de dois 1's, possivelmente com excepção do 0 mais à direita. A expressão regular

$$(1 + 011)^\star$$

define a linguagem das palavras que têm pelo menos dois 1's imediatamente à direita de cada 0.

Exemplo 38 A linguagem das palavras que têm *exatamente* dois 1's entre cada duas ocorrências consecutivas de 0's é definida pela expressão regular

$$1^\star + 1^\star 0(110)^\star 1^\star$$

a qual é equivalente à expressão $1^\star(\varepsilon + 0(110)^\star 1^\star)$. De facto, se x é uma palavra com exatamente dois 1's entre cada duas ocorrências consecutivas de 0, então se x for da forma

$$x_1 0 x_2$$

sendo o 0 indicado o 0 mais à esquerda, temos $x_1 \in \mathcal{L}(1^\star)$ e ou não há 0's em x_2 (isto é, $x_2 \in \mathcal{L}(1^\star)$) ou $x_2 = 110x_3$, com x_3 pertencente exatamente à mesma linguagem que x_2. Assim conclui-se que $x_2 \in \mathcal{L}((110)^\star 1^\star)$.

Exemplo 39 Consideremos ainda mais alguns exemplos.

- A linguagem das palavras em $\{a,b\}^\star$ que têm aa como prefixo e bbb como sufixo é $\{aa\}\{a,b\}^\star\{bbb\}$, sendo descrita, por exemplo, pela expressão regular $aa(a + b)^\star bbb$.
- A linguagem $\{ab^n a \mid n \in \mathbb{N}, n \geq 3\}$ de alfabeto $\{a,b\}$ é descrita pela expressão regular $abbbb^\star a$.
- A linguagem $\{0b^{3n}0^{2m} \mid n,m \in \mathbb{N}\}$ de alfabeto $\{0,b\}$ é descrita pela expressão regular $0(bbb)^\star(00)^\star$.

  Não há relação entre m e n. Se a linguagem fosse $\{0b^{3n}0^{2n} \mid n \in \mathbb{N}\}$, não seria uma linguagem regular, como veremos adiante. Nesta última, o número de 0's na palavra excede numa unidade dois terços do número de b's.
- A linguagem das palavras em $\{a,b\}^\star$ que têm número par de a's é $(\{a\}\{b\}^\star\{a\} \cup \{b\})^\star$, sendo descrita, por exemplo, pela expressão regular

  $$(ab^\star a + b)^\star$$

  Observemos que se uma palavra tiver número par de a's e esse número não for zero, então a palavra pode ser partida em blocos sendo cada bloco uma sequência de b's ou uma palavra de $\mathcal{L}(ab^\star a)$.

Por exemplo, para babbbbbabbbbbbbbbaaaa tem-se a seguinte decomposição

$$\begin{array}{ccccccc} \underbrace{\texttt{b}} & & \underbrace{\texttt{abbbbba}} & \underbrace{\texttt{bbbbbbbbb}} & & \underbrace{\texttt{aa}} & \quad\underbrace{\texttt{aa}} \\ \in & & \in & \in & & \in & \in \\ \mathcal{L}(\texttt{b}^\star) & & \mathcal{L}(\texttt{ab}^\star\texttt{a}) & \mathcal{L}(\texttt{b}^\star) & & \mathcal{L}(\texttt{ab}^\star\texttt{a}) & \mathcal{L}(\texttt{ab}^\star\texttt{a}) \end{array}$$

- A linguagem das palavras em $\{\texttt{a},\texttt{b}\}^\star$ que têm número par de a's ou ímpar de b's é regular, pois é descrita, por exemplo, pela expressão

  $$(\texttt{ab}^\star\texttt{a}+\texttt{b})^\star+(\texttt{ba}^\star\texttt{b}+\texttt{a})^\star\texttt{ba}^\star$$

- A linguagem das palavras de alfabeto $\{\texttt{0},\texttt{1}\}$ que têm 00 como subpalavra é $\{\texttt{0},\texttt{1}\}^\star\{\texttt{00}\}\{\texttt{0},\texttt{1}\}^\star$ e é descrita pela expressão regular

  $$(\texttt{0}+\texttt{1})^\star\texttt{00}(\texttt{0}+\texttt{1})^\star$$

  Uma expressão regular equivalente (ou seja, que descreve exatamente a mesma linguagem) é

  $$(\texttt{01}+\texttt{1})^\star\texttt{00}(\texttt{0}+\texttt{1})^\star$$

  pois a expressão regular $(\texttt{01}+\texttt{1})^\star$ descreve as palavras em $\{\texttt{0},\texttt{1}\}^\star$ que, se tiverem 0's, então têm um 1 imediatamente à direita de cada 0. Assim, ao definirmos $(\texttt{01}+\texttt{1})^\star\texttt{00}(\texttt{0}+\texttt{1})^\star$ como sendo o padrão das palavras que têm 0's consecutivos, o par de 0's que "destacámos" é o par de 0's consecutivos mais à esquerda na palavra.

- A linguagem das palavras em $\{\texttt{0},\texttt{1}\}^\star$ que não têm 0's consecutivos pode ser descrita por $(\texttt{01}+\texttt{1})^\star(\texttt{0}+\varepsilon)$.

3.2 Autómatos finitos

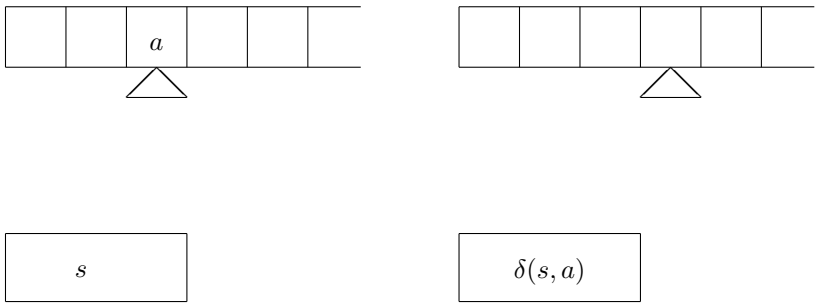
Um *reconhecedor de uma linguagem L sobre Σ* é uma máquina que, dada uma palavra x de $\Sigma^\star$, resolve o problema de decidir se x é palavra de L ou não. Vamos ver que os autómatos finitos constituem um modelo abstrato de máquinas com capacidade para reconhecimento de linguagens regulares (e apenas linguagens regulares).

3.2.1 Autómatos finitos determinísticos

Um **autómato finito determinístico (AFD)** é um modelo de um sistema cujo comportamento pode ser descrito como uma sequência de acontecimentos (estímulo-resposta) no tempo. Num dado momento, o sistema encontra-se em alguma das suas *configurações internas*, que são em **número finito**. Tais configurações, ditas **estados**, funcionam como memória sobre passos anteriores e permitem determinar o comportamento do sistema para estímulos subsequentes. Num AFD, em cada estado, cada estímulo (i.e., **símbolo de entrada**) determina um e um só estado posterior, ou seja,

tem um efeito **determinístico**: o estado seguinte é uma função (no sentido matemático) do estado atual e do símbolo de entrada. A sequência de configurações anteriores a um dado estímulo constitui a história do sistema. Os estados armazenam apenas o que for relevante dessa história.

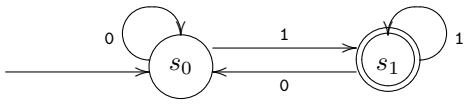
Assim, um autómato finito determinístico pode ser visto como um modelo de uma máquina com um conjunto finito de estados — **máquina finita** — a que se fornece uma fita, onde está escrita uma palavra x de $\Sigma^\star$ que se pretende analisar, para decidir se pertence ou não à linguagem que a máquina carateriza. A máquina tem uma cabeça de leitura, colocada inicialmente de forma a ler o símbolo de x mais à esquerda, e um mecanismo de transição de estados. Num dado estado s, lê o símbolo na fita que se está na posição em que se encontra a cabeça de leitura e, sendo a tal símbolo, passa ao estado $\delta(s, a)$ e move a cabeça de leitura para a posição seguinte (à direita). A máquina **aceita** ou **reconhece** a sequência x dada na fita se, quando acabar de a processar, estiver num *estado final* (estado de aceitação).



Definição 2 *Um* **autómato finito determinístico (AFD)** $\mathcal{A}$ *é descrito por um quinteto* $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$*, sendo* S *o conjunto de* **estados***, que é finito,* Σ *o* **alfabeto** *de símbolos de entrada,* s_0 *o* **estado inicial** *do autómato,* $F \subseteq S$ *o conjunto de* **estados finais***, e* δ *uma função de* $S \times \Sigma$ *em* S*, designada por* **função de transição**.

A linguagem **aceite** ou **reconhecida** pelo autómato $\mathcal{A}$ é o conjunto das palavras de $\Sigma^\star$ que $\mathcal{A}$ aceita. Denota-se por $\mathcal{L}(\mathcal{A})$. Assim, dizemos que uma *linguagem* L *é aceite pelo autómato* $\mathcal{A}$ se e só se $L = \mathcal{L}(\mathcal{A})$.

Exemplo 40 O esquema abaixo representa o **diagrama de transição** de um AFD que tem dois estados, s_0 e s_1. As **transições**, representadas pelas "setas" (arcos), são: do estado s_0 com `0` para o estado s_0, do estado s_0 com `1` para o estado s_1, do estado s_1 com `1` para s_1 e de s_1 com `0` para s_0.



A fita não é representada. Cada palavra a analisar é processada da esquerda para a direita. O **estado inicial** é aquele em que o autómato se encontra quando começa a processar a palavra dada e é apontado por uma seta (sem origem), sendo s_0. O estado s_1, assinalado por duas circunferências, é **estado final**. O autómato **aceita** a palavra dada se depois

de processar todos os seus símbolos estiver em s_1. Neste caso, as sequências em $\{0,1\}^\star$ que são aceites pelo autómato são todas e apenas as que terminam em 1, ou seja, $\mathcal{L}(\mathcal{A}) = \{0,1\}^\star\{1\}$.

Representações da função de transição

- A função de transição $\delta : S \times \Sigma \to S$ pode ser definida com a indicação de $\delta(s,a)$, para cada $s \in S$ e $a \in \Sigma$.

  Exemplo: Seja $\mathcal{A} = (\{s_0,s_1\},\{0,1\},\delta,s_0,\{s_1\})$ um AFD, em que $\delta(s_0,0) = s_0$, $\delta(s_0,1) = s_1$, $\delta(s_1,1) = s_1$, e $\delta(s_1,0) = s_0$.

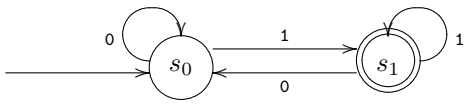
- A função de transição é uma relação binária de $S \times \Sigma$ em S, podendo ser representada por um conjunto de pares $((s,a),s')$, com $s,s' \in S$, $a \in \Sigma$. Para facilitar a notação pode-se utilizar ternos (s,a,s').

  Exemplo: Seja $\mathcal{A} = (\{s_0,s_1\},\{0,1\},\delta,s_0,\{s_1\})$ um autómato finito determinístico, em que δ é definida por $\delta = \{(s_0,0,s_0),(s_0,1,s_1),(s_1,1,s_1),(s_1,0,s_0)\}$.

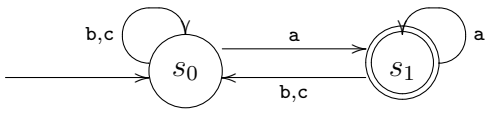
- Pode ser conveniente representar a função de transição por uma tabela. Por exemplo, para o mesmo AFD $\mathcal{A}$ seria:

	0	1
s_0	s_0	s_1
s_1	s_0	s_1

- Um autómato finito pode ser representado pelo seu **diagrama de transição**, que é uma representação baseada num multigrafo dirigido com símbolos associados aos arcos (ramos). Os vértices identificam os estados do autómato. À transição do estado s para o estado s' pelo símbolo de entrada a corresponde o arco do nó s para o nó s', etiquetado por a. O diagrama de transição do AFD indicado é:

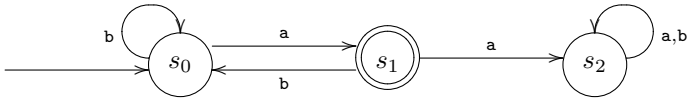


  Por convenção, os estados finais são denotados por duas circunferências e o estado inicial é apontado por uma seta. Para não sobrecarregar a figura, sempre que existir mais do que um arco com a mesma origem e destino, para vários símbolos do alfabeto, representamos apenas um arco no diagrama mas etiquetado com tais símbolos separados por vírgulas. Por exemplo, o autómato seguinte reconhece a linguagem de alfabeto $\Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c}\}$ constituída pelas palavras que terminam em a.

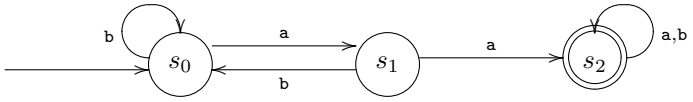


Exemplo 41 Consideramos agora mais alguns exemplos.

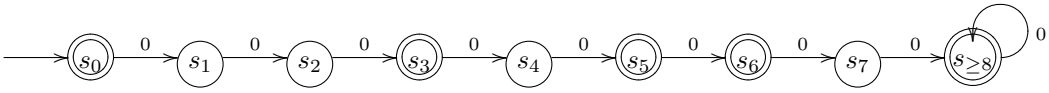
- A linguagem de alfabeto $\{\mathtt{a}, \mathtt{b}\}$ constituída pelas palavras que terminam em a e não têm a's consecutivos é a linguagem reconhecida pelo autómato seguinte.



- A linguagem $\{x \mid x \in \{\mathtt{a}, \mathtt{b}\}^\star$ e tem aa como subpalavra$\}$ é reconhecida pelo autómato:

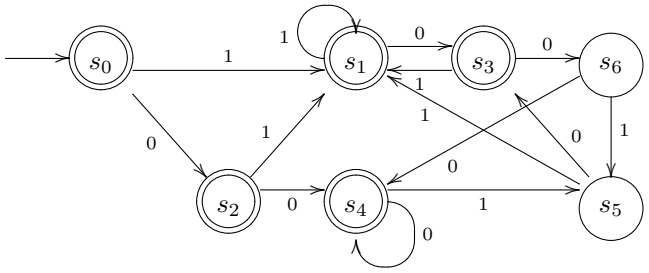


- O autómato finito determinístico que aceita $\{\mathtt{000}, \mathtt{00000}\}^\star$, sobre $\Sigma = \{\mathtt{0}\}$, e tem menor número de estados é:



As designações escolhidas para os estados indicam o número de 0's lidos.

- O AFD mínimo que reconhece $\{x \mid x \in \{\mathtt{0}, \mathtt{1}\}^\star$ e x não termina em 100 nem em 001$\}$ é o seguinte:



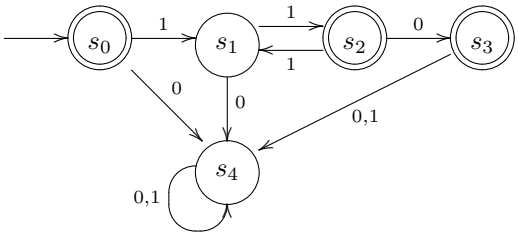
A menos das designações dos estados, qualquer AFD que reconheça esta linguagem tem maior número de estados do que este. Como justificar a necessidade de cada uma destas transições? O teorema de Myhill-Nerode, que estudaremos mais à frente, permite-nos responder a esta questão. Contudo, podemos tentar explicar o **que memoriza cada estado** sobre o prefixo lido (que pode ser já a palavra completa) e desse modo justificar as transições definidas. Assim, temos:

s_0: a palavra é ε;
s_1: a palavra termina em 1 mas não em 001;
s_3: a palavra termina em 10;
s_4: a palavra termina em 00 mas não em 100;
s_5: a palavra termina em 001;
s_6: a palavra termina em 100;
s_2: a palavra é 0;

– Se o autómato consumisse 1 no estado inicial s_0 e se mantivesse em s_0, então, para palavras que começam por 1, atuaria como se os primeiros 1's não fossem relevantes. Se não memorizar o facto de o primeiro 1 poder ser o início de 100, não consegue distinguir 100 de 00. Mas, tem de o fazer pois $\mathtt{100} \notin L$ e $\mathtt{00} \in L$. Portanto, s_1 não é equivalente ao estado s_0.

– Se o autómato consumisse 0 em s_0 e se mantivesse em s_0, então, para palavras que começam por 0, não memorizava o facto de o primeiro 0 poder ser o início de 001. Não distinguiria 001 de 00, mas tem de o fazer pois $001 \notin L$ de $00 \in L$. Portanto, s_0 e s_2 têm de ser distintos. E, s_2 e s_1 também têm de ser distintos para que possa distinguir 100 de 000. De modo semelhante, podemos justificar a existência de cada um dos restantes estados: o que cada estado memoriza é, de certo modo, o sufixo da palavra lida que pode ser determinante para a aceitação ou rejeição da palavra.

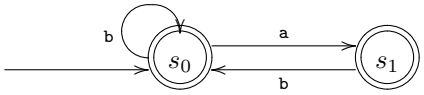
- A linguagem $\{(11)^n \mid n \geq 0\} \cup \{(11)^n 0 \mid n \geq 1\}$ é a linguagem reconhecida, por exemplo, pelo autómato



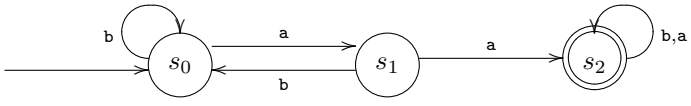
e pode ser descrita pela expressão regular $\varepsilon + 11(11)^\star + 11(11)^\star 0$, a qual é equivalente a $(11)^\star + 11(11)^\star 0$.

Função de transição total versus parcial. Se a função de transição δ pudesse não ser total, isto é, pudesse não estar definida para alguns pares $(s, a) \in S \times \Sigma$, o autómato que se obteria (e que não seria um AFD), podia **encravar** para algumas palavras em $\Sigma^\star$. *Encravar* no sentido de não conseguir acabar de ler/processar a palavra, por não haver transições por certos símbolos num dado estado. Nesse caso, as palavras que fizessem o autómato encravar seriam consideradas como **não aceites**. Não vamos considerar autómatos desse tipo, excepto nos Exemplos 42 e 43, embora diagramas semelhantes possam surgir como representações de AFNDs, que abordaremos na próxima secção.

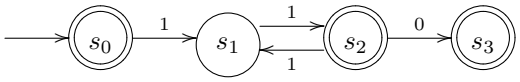
Exemplo 42 A linguagem das palavras em $\{\mathtt{a}, \mathtt{b}\}^\star$ que não têm aa como subpalavra é a linguagem reconhecida pelo autómato $\mathcal{A}$ seguinte (que não é um AFD).



Este autómato encrava se a palavra tiver aa como subpalavra. Se quisessemos representar o autómato que reconhecia $\Sigma^\star \setminus \mathcal{L}(\mathcal{A}) = \{x\mathtt{aa}y \mid x, y \in \{\mathtt{a}, \mathtt{b}\}^\star\}$, isto é, a linguagem das palavras que têm aa como subpalavra, então o "estado" que anteriormente omitimos seria relevante.



Exemplo 43 A linguagem $\{(\texttt{11})^n \mid n \geq 0\} \cup \{(\texttt{11})^n\texttt{0} \mid n \geq 1\}$ é a linguagem reconhecida, por exemplo, pelo autómato seguinte, onde δ é uma função parcial (não o entendemos como um AFD).



Tal autómato encrava se a palavra começar por `0` ou se tiver um `0` depois de um bloco (maximal) de `1`'s em número ímpar, ou ainda se tiver algum `0` ou algum `1` a seguir a um `0`.

Formalização da noção de mudança de configuração num AFD

Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD. Chamamos **configuração** a um par $(s, x) \in S \times \Sigma^\star$, em que s é o estado do autómato e x é a sequência que falta ver num dado momento. A noção de mudança de configuração (por uma transição) pode ser traduzida por uma relação binária $\vdash_{\mathcal{A}}$, definida em $S \times \Sigma^\star$ por $(s, x) \vdash_{\mathcal{A}} (s', x')$ se e só se $x = ax'$ e $s' = \delta(s, a)$, com $a \in \Sigma$, para $s, s' \in S$ e $x, x' \in \Sigma^\star$. Ou seja,

$$\forall s \in S \;\; \forall x \in \Sigma^\star \;\; \forall a \in \Sigma \quad (\; (s, ax) \vdash_{\mathcal{A}} (s', x) \quad sse \quad \delta(s, a) = s').$$

Assim, informalmente, $\vdash_{\mathcal{A}}$ traduz a mudança de configuração determinada por **uma** transição, definindo o novo estado do autómato e o que falta ver da sequência que lhe foi dada, após tal transição. Sempre que não houver ambiguidade quanto ao autómato que está a ser referido, escrevemos simplesmente $\vdash$.

A relação de mudança de configuração em i passos é denotada por $\vdash^i_{\mathcal{A}}$ e pode ser entendida como o resultado da composição de $\vdash_{\mathcal{A}}$ consigo mesma, i vezes. Recordamos sendo R e S relações binárias definidas num conjunto B, a relação RS, a relação composta de R com S, é dada por $RS = \{(x, y) \mid (x, z) \in R \text{ e } (z, y) \in S, \text{ para algum } z \in B\}$. No caso de as relações R e S serem funções, esta noção coincide com a composição de funções. Assim, $\vdash^2_{\mathcal{A}} = \vdash_{\mathcal{A}} \vdash_{\mathcal{A}}$ e $\vdash^i_{\mathcal{A}} = \vdash^{i-1}_{\mathcal{A}} \vdash_{\mathcal{A}}$, para todo $i \geq 2$. Apesar da colisão de notações, não confundamos com a concatenação de linguagens.

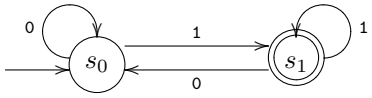
Num AFD, para cada configuração (s, x), com $x \in \Sigma^\star \backslash \{\varepsilon\}$, existe um e um só (s', x') tal que $(s, x) \vdash_{\mathcal{A}} (s', x')$, pois δ é uma função de $S \times \Sigma$ em S. De modo análogo, para qualquer $i \in \mathbb{N}$, se existir (s', x') tal que $(s, x) \vdash^i_{\mathcal{A}} (s', x')$ então (s', x') é único. E, (s', x') existe se e só se $i \leq |x|$. É neste sentido, que o autómato se diz determinístico. O autómato preserva a sua configuração se não realizar qualquer transição, pelo que, $\vdash^0_{\mathcal{A}}$ é a relação de identidade.

O fecho reflexivo e transitivo da relação $\vdash_{\mathcal{A}}$, denotado por $\vdash^\star_{\mathcal{A}}$ (ou por $\vdash^\star$, se não houver mais do que um autómato no contexto), traduz a *alteração de configuração após zero ou mais transições*, em número finito. O fecho reflexivo e transitivo pode ser calculado como o fecho reflexivo do fecho transitivo, pelo que $\vdash^\star = I_{S\times\Sigma^\star} \cup \bigcup_{i\in\mathbb{N}} \vdash^i$, onde $I_{S\times\Sigma^\star}$ denota a relação identidade (ou seja, $\vdash^0$). Formalmente, a linguagem aceite pelo AFD $\mathcal{A}$ é

$$\mathcal{L}(\mathcal{A}) = \{x \mid x \in \Sigma^\star \text{ e existe } f \in F \text{ tal que } (s_0, x) \vdash^\star_{\mathcal{A}} (f, \varepsilon)\}$$

o que significa que $x \in \mathcal{L}(\mathcal{A})$ se e só se x leva o autómato do estado inicial s_0 a algum estado final f, sendo totalmente consumida.

Exemplo 44 Consideremos o autómato finito determinístico representado por:



Aspalavras `01`, `1101` e `1111` são aceites por este AFD e ε, `00`, `11110` não são aceites, pois:

$$\begin{array}{llll} (s_0, \texttt{01}) & \vdash^2 & (s_1, \varepsilon) & \qquad (s_0, \varepsilon) \quad \vdash^\star \quad (s_0, \varepsilon) \\ (s_0, \texttt{1101}) & \vdash^4 & (s_1, \varepsilon) & \qquad (s_0, \texttt{00}) \quad \vdash^2 \quad (s_0, \varepsilon) \\ (s_0, \texttt{1111}) & \vdash^4 & (s_1, \varepsilon) & \qquad (s_0, \texttt{11110}) \quad \vdash^5 \quad (s_0, \varepsilon) \end{array}$$

Por exemplo, $(s_0, \texttt{11110}) \vdash^5 (s_0, \varepsilon)$ porque:

$$(s_0, \texttt{11110}) \vdash (s_1, \texttt{1110}) \vdash (s_1, \texttt{110}) \vdash (s_1, \texttt{10}) \vdash (s_1, \texttt{0}) \vdash (s_0, \varepsilon)$$

A linguagem aceite pelo autómato é o conjunto das palavras em $\{\texttt{0}, \texttt{1}\}^\star$ que terminam em `1`.

Prova formal de que o autómato representado reconhece $\{\texttt{0}, \texttt{1}\}^\star\{\texttt{1}\}$. Para poder relacionar $\vdash^{n+1}$ com $\vdash^n$, é importante caraterizar não somente as sequências que levam o autómato a s_1 mas também a s_0. Ou seja, provar que, para $n \in \mathbb{N}$, $x \in \Sigma^n$ e $y \in \Sigma^\star$, se tem $(s_0, xy) \vdash^n (s, y)$ se e só se ou x termina em `1` e $s = s_1$ ou x não termina em `1` e $s = s_0$. Aqui, xy representa a palavra inicial e x o prefixo que será consumido nas n primeiras transições.

Prova (por indução sobre n):

- (Caso de base) Para $n = 0$, tem-se $x = \varepsilon$ e $(s_0, \varepsilon y) \vdash^0 (s, y)$ sse $s = s_0$, por definição de $\vdash_0$, sendo verdade que ε não termina em `1` e $s = s_0$.

- (Hereditariedade) Supomos, como hipótese de indução, que a condição se verifica para n fixo. Seja $w \in \Sigma^{n+1}$ e $z \in \Sigma^\star$, quaisquer. Então, $w = xa$, para algum $a \in \Sigma$ e $x \in \Sigma^n$, e $(s_0, (xa)z) \vdash^{n+1} (s, z)$ se e só se $(s_0, x(az)) \vdash^n (s', az) \vdash (s, z)$, por definição de AFD e de $\vdash^{n+1}$ (mudança de configuração por $n + 1$ transições). Mas, por hipótese de indução, sabemos que $(s_0, x(az)) \vdash^n (s', az)$ se e só se ou $s' = s_1$ e x não termina em `1` ou $s' = s_0$ e x não termina em `1`. É necessário considerar agora quatro casos para $(s', az) \vdash (s, z)$ pois, além de $s' \in \{s_0, s_1\}$, podemos ter $a \in \{\texttt{0}, \texttt{1}\}$. De acordo com a definição de $\vdash$ (alteração de configuração por uma transição), podemos concluir que $s = s_0$ se $a = \texttt{0}$, e que $s = s_1$ se $a = \texttt{1}$. Mas, como $w = xa$, tal significa que se w termina em `1` então $s = s_1$ e se w não termina em `1` então $s = s_0$.

Portanto, por indução matemática, concluimos que a condição se verifica para todo $n \in \mathbb{N}$. Como corolário da proposição demonstrada, temos $(s_0, x) \vdash^n (s_1, \varepsilon)$, para algum $n \geq 0$, se e só se ou x termina em `1`. Portanto, $\mathcal{L}(A)$ é o conjunto das palavras que terminam em `1`. □

A relação $\vdash^\star$ não é uma função em $S \times \Sigma^\star$. Contudo, o determinismo do AFD implica a unicidade da configuração que resulta de uma sequência de transições e, em particular, da configuração após consumir a palavra dada. Tal resultado é formalizado pela Proposição 10, podendo ser usado para simplificar a prova de que uma dada palavra não é aceite por um dado autómato.

Proposição 10 *Se $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ é um autómato finito determinístico então, qualquer que seja $(s, x) \in S \times \Sigma^\star$, existe um e um só $s' \in S$ tal que $(s, x) \vdash^\star (s', \varepsilon)$.*

Prova: Por indução sobre $|x|$.

(Caso de base) Se $|x| = 0$ então $x = \varepsilon$, e $(s, \varepsilon) \vdash^\star (s', \varepsilon)$ sse $s' = s$, pois o AFD mantém a configuração (não tem transições-ε).

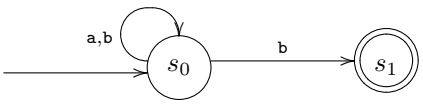
(Hereditariedade) Como hipótese de indução, supomos que, para $k \in \mathbb{N}$ fixo, e para qualquer $x \in \Sigma^k$ (isto é, com $|x| = k$), se tem $\forall s \in S\ \exists^1 s' \in S\ \ (\ (s, x) \vdash^\star (s', \varepsilon))$. Aqui, $\exists^1$ lê-se "existe um e um só". E, vamos mostrar que, se w é uma palavra qualquer em Σ^{k+1} (ou seja, $|w| = k + 1$), então, para qualquer $q \in S$, existe um e um só q' tal que $(q, w) \vdash^\star (q', \varepsilon)$. Ora, se $|w| = k + 1$ então $w = aw'$ para algum $a \in \Sigma$ e $w' \in \Sigma^\star$. Como $|w'| = |w| - 1$, concluimos que $|w'| = k$, e, portanto, se necessário, podemos aplicar a hipótese de indução a w'. Por definição de $\vdash$, existe um só $q'' \in S$ e um só $v \in \Sigma^\star$ tal que $(q, aw') \vdash (q'', v)$, nomeadamente $q'' = \delta(q, a)$ e $v = w'$. E, pela hipótese, existe um só $q' \in S$ tal que $(q'', w') \vdash^\star (q', \varepsilon)$. Logo, $(q, aw') \vdash (q'', v) \vdash^\star (q', \varepsilon)$, para q'', v e q' únicos, e portanto $(q, w) \vdash^\star (q', \varepsilon)$, para $q' \in S$ único.

Portanto, pelo princípio de indução, o resultado enunciado é válido. □

Corolário 10.1 *Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD e seja $x \in \Sigma^\star$. Então, $x \notin \mathcal{L}(\mathcal{A})$ sse $(s_0, x) \vdash^\star (s, \varepsilon)$ e $s \notin F$.*

3.2.2 Autómatos finitos não deterministicos

Consideremos o diagrama de transição seguinte.



Neste caso, se o autómato estiver no estado s_0 e consumir `b`, pode manter-se em s_0 ou passar ao estado s_1. É neste sentido que se diz *não determinístico*. Se admitirmos que o autómato aceita uma dada palavra se esta o puder levar do estado inicial a algum dos estados finais, concluimos que o autómato anterior define a linguagem descrita pela expressão $(\mathtt{a} + \mathtt{b})^\star\mathtt{b}$.

Definição 3 *Um* **autómato finito não determinístico (AFND)** $\mathcal{A}$ *é descrito por um quinteto* $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$*, sendo* S *o conjunto* finito *de estados,* Σ *o alfabeto de símbolos de entrada,* s_0 *o estado inicial do autómato,* $F \subseteq S$ *o conjunto de estados finais e* δ *a função de transição, a qual é uma função de* $S \times \Sigma$ *em* 2^S*.*

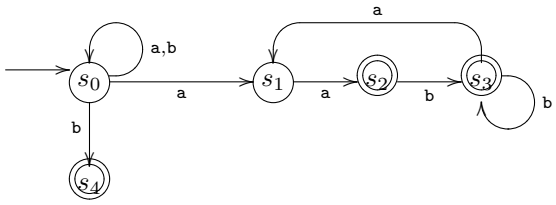
Recordamos que 2^S denota o conjunto dos subconjuntos de S.

Exemplo 45 O diagrama de transição apresentado acima corresponde ao AFND $\mathcal{A} = (\{s_0, s_1\}, \{\mathtt{a}, \mathtt{b}\}, \delta, s_0, \{s_1\})$, com $\delta(s_0, \mathtt{a}) = \{s_0\}$, $\delta(s_0, \mathtt{b}) = \{s_1\}$ e $\delta(s_1, \mathtt{a}) = \delta(s_1, \mathtt{b}) = \{\ \}$.

Exemplo 46 O AFND $\mathcal{A} = (\{s_0, s_1, s_2, s_3, s_4\}, \{\mathtt{a}, \mathtt{b}\}, \delta, s_0, \{s_2, s_3, s_4\})$, com δ dada por

$$\begin{array}{lllll} \delta(s_0, \mathtt{a}) = \{s_0, s_1\} & \delta(s_1, \mathtt{a}) = \{s_2\} & \delta(s_2, \mathtt{a}) = \{\} & \delta(s_3, \mathtt{a}) = \{s_1\} & \delta(s_4, \mathtt{a}) = \{\} \\ \delta(s_0, \mathtt{b}) = \{s_0, s_4\} & \delta(s_1, \mathtt{b}) = \{\} & \delta(s_2, \mathtt{b}) = \{s_3\} & \delta(s_3, \mathtt{b}) = \{s_3\} & \delta(s_4, \mathtt{b}) = \{\} \end{array}$$

pode ser representado pelo diagrama representado abaixo.



Este autómato reconhece a linguagem definida pela expressão regular $(\mathtt{a} + \mathtt{b})^\star(\mathtt{b} + \mathtt{a}(\mathtt{abb}^\star\mathtt{a})^\star(\mathtt{a} + \mathtt{abb}^\star))$.

Formalização da noção de mudança de configuração num AFND. Como no caso dos AFDs, podiamos definir uma configuração como um par (s, x), com $s \in S$ e $x \in \Sigma^\star$, onde x indica o estado em que o autómato se encontra e x a palavra a processar. A relação de mudança de configuração num passo, $\vdash_{\mathcal{A}}$, seria traduzida por uma relação binária definida em $S \times \Sigma^\star$. Teriamos, $(s, ay) \vdash_{\mathcal{A}} (s', y)$ se e só se $s' \in \delta(s, a)$, para $a \in \Sigma$ e $y \in \Sigma^\star$. A linguagem aceite por $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ seria o conjunto das palavras $x \in \Sigma^\star$ tais que $(s_0, x) \vdash^\star_{\mathcal{A}} (f, \varepsilon)$, para algum $f \in F$.

Alternativamente, a noção de **configuração para um AFND** pode ser formalizada por um par (E, x), em que E é o conjunto de todos os estados em que o autómato pode estar num dado instante e x é a sequência que falta ver. Deste modo, em cada instante *há uma só configuração* para o AFND (como a Proposição 11 refere). A relação de mudança de configuração num passo, $\vdash_{\mathcal{A}}$, é uma relação binária em $2^S \times \Sigma^\star$ e é definida por: quaisquer que sejam $E, E' \in 2^S$ e $x, x' \in \Sigma^\star$,

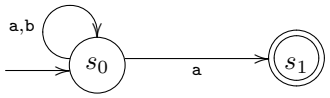
$$(E, x) \vdash_{\mathcal{A}} (E', x') \quad sse \quad x = ax' \text{ para algum } a \in \Sigma \text{ e } \ E' = \bigcup_{s \in E} E_s, \text{ em que } E_s = \delta(s, a).$$

Assim, $(E, ay) \vdash_{\mathcal{A}} (E', y)$ se e só se $E' = \bigcup_{s \in E} \ \delta(s, a)$. Com esta definição, a noção de linguagem aceite pelo autómato é $\mathcal{L}(A) = \{x \mid (\{s_0\}, x) \vdash^\star_{\mathcal{A}} (E, \varepsilon) \ \wedge \ E \cap F \neq \emptyset\}$. Esta é a noção usada a seguir, na Proposição 11.

Proposição 11 *Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFND. Então, qualquer que seja $(E, x) \in 2^S \times \Sigma^\star$, existe um e um só $E' \in 2^S$ tal que $(E, x) \vdash^\star (E', \varepsilon)$. Logo, $x \notin \mathcal{L}(\mathcal{A})$ se e só se $(\{s_0\}, x) \vdash^\star (E', \varepsilon)$ e $E' \cap F = \emptyset$.*

Prova: Por indução sobre $|x|$ e analogia com a prova da Proposição 10. □

Exemplo 47 Seja $\mathcal{A} = (\{s_0, s_1\}, \{\mathtt{a}, \mathtt{b}\}, \delta, s_0, \{s_1\})$ um AFND, com $\delta(s_0, \mathtt{a}) = \{s_0, s_1\}$, $\delta(s_1, \mathtt{a}) = \emptyset$, $\delta(s_1, \mathtt{b}) = \emptyset$, e $\delta(s_0, \mathtt{b}) = \{s_0\}$, o qual é representado pelo diagrama seguinte.



Tem-se

$$\begin{array}{rcll} (\{s_0\}, \mathtt{aba}) & \vdash & (\{s_0, s_1\}, \mathtt{ba}) & (\text{por } \delta(s_0, \mathtt{a}) = \{s_0, s_1\}) \\ & \vdash & (\{s_0\}, \mathtt{a}) & (\text{por } \delta(s_0, \mathtt{b}) = \{s_0\} \text{ e } \delta(s_1, \mathtt{b}) = \emptyset) \\ & \vdash & (\{s_0, s_1\}, \varepsilon) & \end{array}$$

Logo, $(\{s_0\}, \mathtt{aba}) \vdash^\star (\{s_0, s_1\}, \varepsilon)$, ou seja $\mathtt{aba} \in \mathcal{L}(\mathcal{A})$, já que s_1 é estado final. Analogamente, podemos verificar que $(\{s_0\}, \mathtt{abab}) \vdash^\star (E, \varepsilon)$ se e só se $E = \{s_0\}$. Portanto, $\mathtt{abab} \notin \mathcal{L}(\mathcal{A})$, já que $\{s_0\} \cap \{s_1\} = \emptyset$.

Os autómatos finitos determinísticos podem ser vistos como casos particulares dos autómatos finitos não determinísticos.

Exemplo 48 O AFD $\mathcal{A} = (\{s_0, s_1\}, \{\mathtt{a}, \mathtt{b}\}, \delta, s_0, \{s_0, s_1\})$, em que $\delta(s_0, \mathtt{b}) = s_0$, $\delta(s_0, \mathtt{a}) = s_1$, $\delta(s_1, \mathtt{b}) = s_0$, e $\delta(s_1, \mathtt{a}) = s_1$, corresponde trivialmente ao AFND $\mathcal{A}' = (\{s_0, s_1\}, \{\mathtt{a}, \mathtt{b}\}, \delta', s_0, \{s_0, s_1\})$, com $\delta'(s_0, \mathtt{b}) = \{s_0\}$, $\delta'(s_0, \mathtt{a}) = \{s_1\}$, $\delta'(s_1, \mathtt{b}) = \{s_0\}$ e $\delta'(s_1, \mathtt{a}) = \{s_1\}$.

A ideia desta construção vai ser usada na prova da Proposição 12, a qual diz que os AFDs não são mais "potentes" que os AFNDs, como reconhecedores de linguagens.

Proposição 12 *Qualquer linguagem que seja reconhecida por um AFD pode ser reconhecida por algum AFND.*

Prova: Seja L uma linguagem tal que $L = \mathcal{L}(\mathcal{A})$ para um AFD $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$. Esse AFD corresponde ao AFND $\mathcal{A}' = (S, \Sigma, \delta', s_0, F)$, em que $\delta'(s, a) = \{\delta(s, a)\}$, para todo $(s, a) \in S \times \Sigma$. É fácil concluir que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. □

O recíproco deste resultado é também válido, como afirma a Proposição 13, permitindo concluir que os AFDs e os AFNDs são igualmente potentes (permitem reconhecer exatamente a mesma classe de linguagens) .

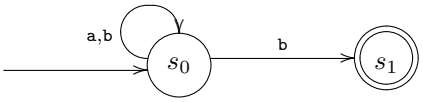
Proposição 13 *Qualquer linguagem que seja reconhecida por um AFND pode ser reconhecida por algum AFD.*

Prova: Dado AFND $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, construimos o AFD $\mathcal{A}' = (2^S, \Sigma, \delta', \{s_0\}, F')$, em que o conjunto de estados finais $F' = \{E \mid E \in 2^S \ \wedge \ E \cap F \neq \{\}\}$ e a função de transição δ' é dada por

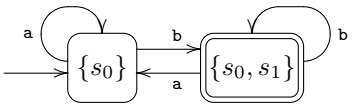
$$\delta'(E, a) \stackrel{\text{def}}{=} \bigcup_{s \in E} \delta(s, a)$$

com $E \in 2^S$ e $a \in \Sigma$ quaisquer. Pode-se mostrar que os autómatos $\mathcal{A}$ e $\mathcal{A}'$ são equivalentes, isto é reconhecem a mesma linguagem (o que segue facilmente das Proposições 10 e 11). □

Esta prova define um método de construção de um AFD equivalente a um AFND dado, que se designa por **método construção baseado em subconjuntos**. Explora o facto de conseguirmos formalizar o comportamento de um autómato não determinístico $\mathcal{A}$ de uma forma "determinística", se dissermos que, em cada passo da análise de uma dada palavra, o autómato $\mathcal{A}$ pode **estar num certo conjunto de estados**. Consideremos, por exemplo, o autómato:

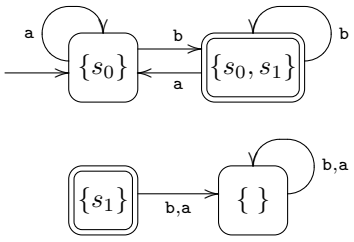


Depois de consumir `ababb`, o conjunto de estados em que o autómato pode estar é $\{s_0, s_1\}$. Do mesmo modo, depois de consumir `ababba`, o conjunto de estados em que o autómato pode estar é $\{s_0\}$. O AFD construído pelo método descrito na prova da Proposição 13, de certa forma, "imita" o autómato finito não determinístico dado. Para o exemplo, tal autómato poderia seria o seguinte, se descartassemos os estados não acessíveis do estado inicial $\{s_0\}$.



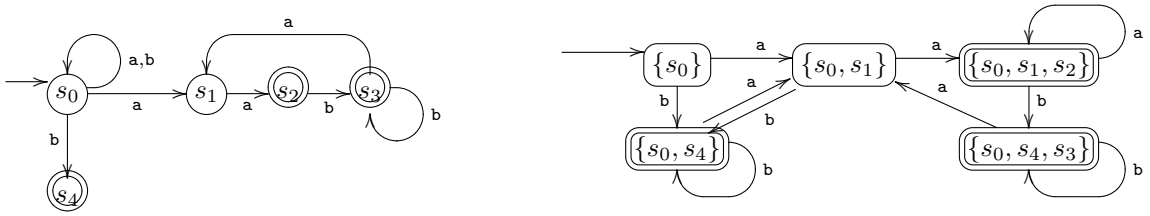
Inicialmente o AFND tem que estar em s_0. O AFD memoriza isso, registando no seu estado inicial que o conjunto de estados possíveis no início para o AFND é $\{s_0\}$. Se estiver no estado s_0 e consumir `b`, então o AFND passa a poder estar em s_0 ou em s_1, o que o AFD memoriza no seu outro estado. Se num dado momento o conjunto de estados em que o AFND pode estar for $\{s_0, s_1\}$ e consumir `b`, então a seguir continua a poder estar em $\{s_0, s_1\}$. Mas, se consumir `a` estando em s_0 ou em s_1, o conjunto de estados em que pode estar no passo seguinte reduz-se a $\{s_0\}$, o que o AFD como "bom imitador" memoriza! Por outro lado, o AFND aceita uma palavra se esta o puder levar do estado inicial a algum dos estados finais. Então, o AFD tem como estados finais todos os que tiverem designações que incluam algum dos estados finais do AFND.

Se seguissemos rigorosamente a prova da Proposição 13, o AFD que se construia para o exemplo anterior teria $2^{\{s_0, s_1\}}$ como conjunto de estados e seria dado pelo diagrama seguinte.



Contudo, neste AFD, os estados $\{\,\}$ e $\{s_1\}$ não são acessíveis do estado inicial $\{s_0\}$, pelo que são trivialmente redundantes. Assim, podemos obter um AFD equivalente se removermos tais estados. Se implementarmos o algoritmo de conversão de AFNDs para AFDs, numa linguagem de programação, devemos considerar como estados do AFD apenas aqueles estados em 2^S acessíveis de $\{s_0\}$, permitindo que, nalgumas instâncias, a construção não seja exponencial em $|S|$, como é no pior caso.

Exemplo 49 Para o AFND $\mathcal{A}$ representado abaixo, à esquerda, o AFD $\mathcal{A}'$ definido na prova da Proposição 13 teria $2^{|S|} = 32$ estados, e seria relativamente grande. Se se partir de $\{s_0\}$ e se considerar apenas os estados que vão sendo encontrados, obtém-se um AFD equivalente com apenas cinco estados (representado à direita).



3.2.3 Autómatos finitos não determinísticos com transições por ε

Os autómatos finitos que introduzimos agora são semelhantes aos autómatos finitos não determinísticos, mas podem ter transições por ε. Ao efetuar uma "transição por ε" (ou transição-ε), o autómato muda de estado sem consumir qualquer símbolo da palavra.

Definição 4 *Um* **automáto finito não determinístico com transições por ε (AFND-ε)** *é descrito por um quinteto $(S, \Sigma, \delta, s_0, F)$ como um AFND, mas δ é uma função de $S \times (\Sigma \cup \{\varepsilon\})$ em 2^S.*

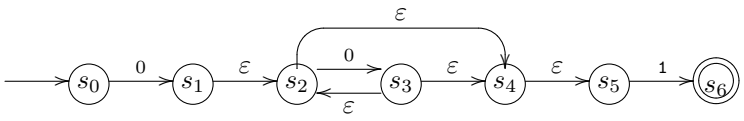
Os AFNDs-ε têm algum interesse pois permitem construir autómatos mais complexos a partir de outros mais simples. Serão fundamentais na prova de que qualquer linguagem regular pode ser reconhecida por um autómato finito

(que estudaremos à frente). A linguagem aceite por um AFND-ε é o conjunto das palavras que levam o autómato do estado inicial a algum estado final.

Denote-se por $\text{Fecho}_\varepsilon(s)$ o conjunto de estados acessíveis do estado s sem consumir qualquer símbolo da palavra, para $s \in S$. Dado $E \in 2^S$, se se imaginar que o autómato pode estar em qualquer estado de E, o conjunto de estados em que o autómato está ou pode vir a estar **sem consumir qualquer símbolo da palavra** e é assim definido:

$$\text{Fecho}_\varepsilon(E) = E \cup \{s' \mid \exists s \in E \quad s' \text{ é acessível de } s \text{ por transições-}\varepsilon\}.$$

Exemplo 50 Seja $\mathcal{A}$ o AFND-ε cujo diagrama de transição é:



A palavra 001 pertence a $\mathcal{L}(\mathcal{A})$, como se conclui, por exemplo, da análise dos percursos seguintes em $\mathcal{A}$:

$$s_0 \xrightarrow{0} s_1 \xrightarrow{\varepsilon} s_2 \xrightarrow{0} s_3 \xrightarrow{\varepsilon} s_4 \xrightarrow{\varepsilon} s_5 \xrightarrow{1} s_6$$
$$s_0 \xrightarrow{0} s_1 \xrightarrow{\varepsilon} s_2 \xrightarrow{0} s_3 \xrightarrow{\varepsilon} s_2 \xrightarrow{\varepsilon} s_4 \xrightarrow{\varepsilon} s_5 \xrightarrow{1} s_6$$

Podemos verificar que $010 \notin \mathcal{L}(\mathcal{A})$, $011 \notin \mathcal{L}(\mathcal{A})$, e que $\mathcal{L}(\mathcal{A}) = \{0^n 1 \mid n \geq 1\} = \mathcal{L}(00^\star 1)$. Os estados acessíveis de cada estado s_i, sem consumir qualquer símbolo da palavra, são dados por

$$\text{Fecho}_\varepsilon(s_0) = \{s_0\} \qquad \text{Fecho}_\varepsilon(s_1) = \{s_1, s_2, s_4, s_5\} \qquad \text{Fecho}_\varepsilon(s_2) = \{s_2, s_4, s_5\}$$
$$\text{Fecho}_\varepsilon(s_3) = \{s_2, s_3, s_4, s_5\} \qquad \text{Fecho}_\varepsilon(s_4) = \{s_4, s_5\} \qquad \text{Fecho}_\varepsilon(s_5) = \{s_5\}$$

e o AFND-ε $\mathcal{A}$ é equivalente ao AFND $\mathcal{A}' = (\{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}, \{0, 1\}, \delta', s_0, \{s_6\})$ em que

$$\delta'(s_0, 0) = \{s_1, s_2, s_4, s_5\} \qquad \delta'(s_1, 0) = \{s_2, s_3, s_4, s_5\} \qquad \delta'(s_2, 0) = \{s_2, s_3, s_4, s_5\}$$
$$\delta'(s_3, 0) = \{s_2, s_3, s_4, s_5\} \qquad \delta'(s_i, 1) = \{s_6\}, \quad 1 \leq i \leq 5$$

com $\delta'(s, a) = \emptyset$, para os restantes $(s, a) \in \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\} \times \{0, 1\}$. O AFND $\mathcal{A}'$ "simula" o comportamento de $\mathcal{A}$, sem efetuar transições-ε, e tem exatamente o mesmo conjunto de estados.

Formalização da noção de mudança de configuração num AFND-ε. Por analogia com o caso dos AFDs, podiamos definir uma configuração como um par (s, x), com $s \in S$ e $x \in \Sigma^\star$, onde x indica o estado em que o autómato se encontra e x a palavra a processar. Como um AFND-ε pode ter transições-ε, a mudança de configuração num passo, $\vdash_{\mathcal{A}}$, seria traduzida por uma relação binária definida em $S \times \Sigma^\star$, em que $(s, ay) \vdash_{\mathcal{A}} (s', y)$ se e só se $s' \in \delta(s, a)$, para $s, s' \in S$, $a \in \Sigma \cup \{\varepsilon\}$ e $y \in \Sigma^\star$. A linguagem aceite por $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ seria o conjunto das palavras $x \in \Sigma^\star$ tais que $(s_0, x) \vdash_{\mathcal{A}}^\star (f, \varepsilon)$, para algum $f \in F$.

Contudo, à semelhança do caso dos AFNDs, algumas provas tornam-se mais simples se se assumir que uma configuração de um AFND-ε é definida por um par (E, x), em que E é um conjunto de estados. Na definição alternativa, que se encontra abaixo, considera-se que, numa transição, o AFND-ε consome um símbolo da palavra dada (se esta não for ε), efetuando as transições-ε necessárias para tal.

$$(E, x) \vdash (E', x') \quad sse \quad \begin{cases} E \neq E' = Fecho_\varepsilon(E) & \text{se } x = x' = \varepsilon \text{ e } E \neq E' \\ E' = Fecho_\varepsilon\left(\bigcup_{s \in Fecho_\varepsilon(E)} \delta(s, a)\right) & \text{se } |x| \geq 1,\ x = ax' \end{cases}$$

Assumimos que $\bigcup_{s \in \emptyset} X_s = \emptyset$. De acordo com esta definição de $\vdash$, a linguagem reconhecida pelo autómato é dada por $\mathcal{L}(\mathcal{A}) = \{x \mid (\{s_0\}, x) \vdash^\star (E, \varepsilon), \text{ para algum } E \in 2^S \text{ tal que } E \cap F \neq \emptyset\}$. Esta definição sugere **o método de conversão de AFNDs-ε para AFDs** descrito na prova da Proposição 14.

Proposição 14 *Qualquer linguagem que seja aceite por um AFND-ε é aceite por algum AFD, e vice-versa.*

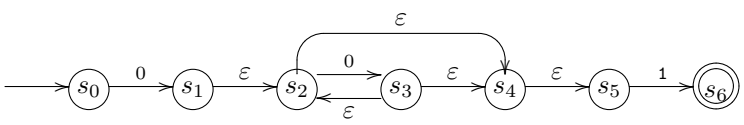
Prova: A prova de que qualquer linguagem que é aceite por um AFD é aceite por um AFND-ε é trivial, uma vez que qualquer AFD pode ser visto como um AFND-ε.

Resta assim, provar a implicação recíproca. Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFND-ε que aceita a linguagem dada. Para definir o AFD equivalente a $\mathcal{A}$, procedemos como na prova da Proposição 13. Se $\mathcal{A}$ estiver no estado s, pode consumir a desde que haja algum arco etiquetado por a com origem em s ou em algum estado acessível de s por transições ε. Mais ainda, depois de consumir esse a, o autómato pode estar em qualquer dos estados que seja acessível por ε do estado a que chegou. Assim o AFD que vamos construir é dado por $\mathcal{A}' = (2^S, \Sigma, \delta', \text{Fecho}_\varepsilon(s_0), F')$ com

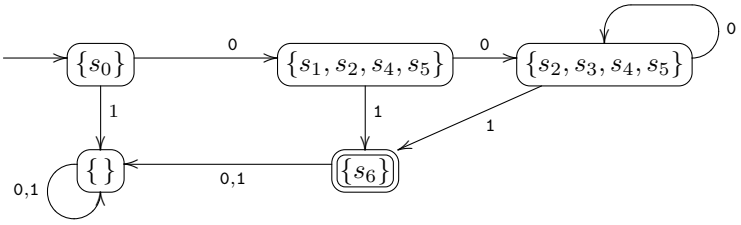
$$\delta'(E, a) = \text{Fecho}_\varepsilon\left(\bigcup_{s \in \text{Fecho}_\varepsilon(E)} \delta(s, a)\right)$$

e $F' = \{E \mid E \in \mathcal{P}(S) \ \wedge \ E \cap F \neq \{\}\}$. Pode-se mostrar que $\mathcal{A}$ e $\mathcal{A}'$ são equivalentes, isto é, reconhecem a mesma linguagem, mas omitimos essa parte da prova. □

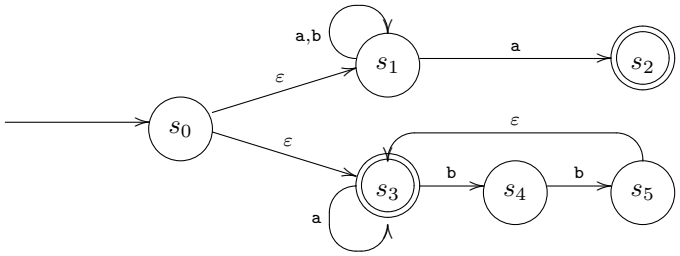
Exemplo 51 Consideremos novamente o AFND-ε representado pelo diagrama de transição seguinte.



Com base na prova da Proposição 14, concluimos que $\mathcal{A}$ é equivalente ao AFD representado abaixo.



Exemplo 52 O autómato representado ilustra uma das vantagens da introdução dos AFNDs-ε. Sendo conhecidos autómatos que aceitam L_1 e L_2, podemos usá-los para construir um autómato que reconhece a união das duas linguagens, $L_1 \cup L_2$. Basta considerar um novo estado inicial donde saem transições-ε para os estados iniciais de cada um desses dois autómatos (que, neste caso, seriam s_1 e s_3).

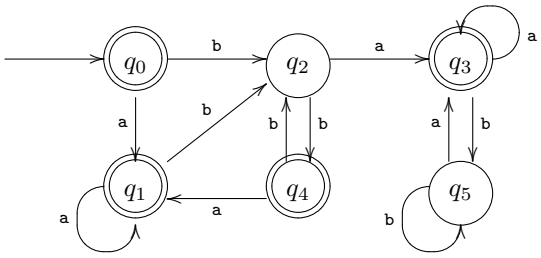


Podemos verificar que, por exemplo, $\mathtt{aa} \in \mathcal{L}(\mathcal{A})$. Partindo do estado inicial, após consumir aa, o conjunto de estados possível é $\{s_1, s_2, s_3\}$, pelo que aa é aceite. Analogamente, depois de consumir bb é $\{s_1, s_3, s_5\}$, pelo que bb é aceite. Como depois de consumir abbab seria $\{s_1, s_4\}$, a palavra abbab não é aceite. Depois de consumir abbaba é $\{s_1, s_2\}$, pelo que abbaba é aceite. Por análise dos percursos que levam a estado final, vemos que a linguagem aceite pelo autómato é descrita pela expressão regular $(\mathtt{a} + \mathtt{b})^\star\mathtt{a} + (\mathtt{bb} + \mathtt{a})^\star$.

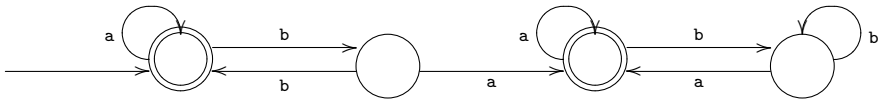
A função de transição do AFD obtido pelo método de conversão, descartando estados não acessíveis do estado inicial $\{s_0, s_1, s_3\}$, é dada por:

$$\delta'(\{s_0, s_1, s_3\}, \mathtt{a}) = \{s_1, s_2, s_3\} = q_1 \qquad \delta'(\{s_0\}, \mathtt{b}) = \{s_1, s_4\} = q_2$$
$$\delta'(\{s_1, s_2, s_3\}, \mathtt{a}) = \{s_1, s_2, s_3\} \qquad \delta'(\{s_1, s_2, s_3\}, \mathtt{b}) = \{s_1, s_4\}$$
$$\delta'(\{s_1, s_4\}, \mathtt{a}) = \{s_1, s_2\} = q_3 \qquad \delta'(\{s_1, s_4\}, \mathtt{b}) = \{s_1, s_3, s_5\} = q_4$$
$$\delta'(\{s_1, s_2\}, \mathtt{a}) = \{s_1, s_2\} \qquad \delta'(\{s_1, s_2\}, \mathtt{b}) = \{s_1\} = q_5$$
$$\delta'(\{s_1, s_3, s_5\}, \mathtt{a}) = \{s_1, s_2, s_3\} \qquad \delta'(\{s_1, s_3, s_5\}, \mathtt{b}) = \{s_1, s_4\}$$
$$\delta'(\{s_1\}, \mathtt{a}) = \{s_1, s_2\} \qquad \delta'(\{s_1\}, \mathtt{b}) = \{s_1\}$$

Portanto, o AFND-ε dado é equivalente ao AFD seguinte:



O AFD obtido não é o **AFD minímo** equivalente ao AFND-ε dado. O AFD minímo, i.e., o que tem menor número de estados, é:

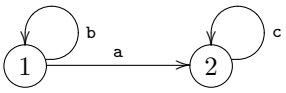


Mais adiante vamos estudar um resultado que nos permite verificar se um dado AFD é ou não é o **AFD minímo** que reconhece uma dada linguagem. Por enquanto, tentemos convencermo-nos, neste caso, da necessidade de cada uma das transições efetuadas, considerando que a linguagem aceite é $\mathcal{L}((\mathtt{a}+\mathtt{b})^\star\mathtt{a}+(\mathtt{bb}+\mathtt{a})^\star)$. Para isso, notemos que, se a palavra dada não pertencer a $\mathcal{L}((\mathtt{bb}+\mathtt{a})^\star)$, tem algum a imediatamente após um bloco maximal de b's em número ímpar e, nesse caso, terá que terminar em a.

3.3 Conversão entre autómatos finitos e expressões regulares

Vamos mostrar que as linguagens regulares (ou seja, as que podem ser descritas por expressões regulares) são as linguagens que podem ser reconhecidas por autómatos finitos.

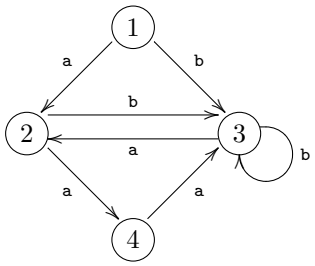
Exemplo 53 Consideremos o diagrama de transição seguinte.



Neste caso é fácil verificar que $\mathtt{b}^\star$ descreve a linguagem determinada por **percursos generalizados** (ou seja, percursos com zero ou mais arcos) de 1 para 1. A linguagem determinada por percursos generalizados de 2 para 2 é descrita por $\mathtt{c}^\star$. Como não existe qualquer percurso de 2 para 1, concluimos que $\emptyset$ descreve a linguagem determinada por percursos de 2 para 1. Finalmente, $\mathtt{b}^\star\mathtt{ac}^\star$ descreve a linguagem determinada por percursos de 1 para 2.

Exemplo 54 No caso seguinte, seria ainda possível descrever por uma expressão regular a linguagens das palavras

que levam o autómato do estado 1 a cada um dos restantes estados?



Os resultados enunciados abaixo, nas Proposições 15 e 16, mostram que as linguagens regulares são as linguagens que podem ser reconhecidas por autómatos finitos. As provas dessas proposições são **provas construtivas** (isto é, apresentam métodos de construção dos *objetos* cuja existência se quer mostrar). Neste caso, os objetos são expressões regulares ou autómatos finitos. Para facilitar a sua compreensão, os métodos referidos – método de Kleene, método eliminação de estados de Brzozowski-McCluskey, e método de método de McNaughton-Yamada-Thompson – só são de facto descritos nas secções seguintes.

Proposição 15 *Qualquer linguagem que seja reconhecida por um autómato finito é regular.*

Prova: Partindo de um autómato finito que reconheça L, obtemos uma expressão regular que descreve L por aplicação do método de Kleene ou do método eliminação de estados de Brzozowski-McCluskey. □

Proposição 16 *Qualquer linguagem regular pode ser reconhecida por um autómato finito.*

Prova: Partindo de uma expressão regular que descreva L, podemos obter um AFND-ε que reconhece L por aplicação do método de McNaughton-Yamada-Thompson. □

3.3.1 Método de Kleene

Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um autómato finito que reconhece uma linguagem L dada. A prova que vamos fazer é válida quer $\mathcal{A}$ seja um AFD, um AFND, ou um AFND-ε. Consideremos o diagrama de transição de $\mathcal{A}$ e suponhamos que os estados são numerados de 1 a $|S|$, sendo 1 o número que se atribui a s_0. A expressão regular que descreve a linguagem determinada por percursos generalizados do estado i para o estado j pode ser assim obtida por recorrência.

Seja $r_{ij}^{(k)}$ a expressão que descreve a sub-linguagem determinada pelos percursos de i para j que passam quando

muito por estados intermédios *etiquetados* com números não superiores a k. Assim,

$$r_{ii}^{(0)} = \begin{cases} \varepsilon & sse \quad \text{não existe qualquer lacete em } i \\ \varepsilon + a_1 \ldots + a_p & sse \quad \text{os lacetes em } i \text{ estão etiquetados com } a_1, \ldots a_p \end{cases}$$

$$r_{ij}^{(0)} = \begin{cases} \emptyset & sse \quad \text{não existe qualquer arco com origem em } i \text{ e fim em } j \\ a_1 + \ldots + a_p & sse \quad a_1, \ldots a_p \text{ etiquetam os arcos com origem em } i \text{ e fim em } j \end{cases}$$

e, para $k \geq 1$, define-se agora $r_{ij}^{(k)}$ por recorrência por

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} + r_{ik}^{(k-1)} (r_{kk}^{(k-1)})^\star r_{kj}^{(k-1)}.$$

Esta definição traduz o facto de os percursos generalizados de i para j que passam quando muito por estados numerados até k serem:

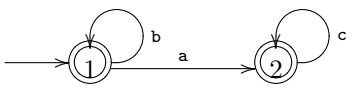
- percursos de i para j que passam quando muito por estados numerados até $k-1$, ou
- poderem ser obtidos por concatenação de um qualquer percurso de i para k que não tenha k como estado intermédio, com um ou nenhum circuito de k para k, e com um percurso qualquer de k para j que não passe por k. E ainda, que cada circuito de k para k se decompõe como uma sequência de circuitos de k para k em que k só ocorre como origem e fim de cada um deles.

Por definição de expressões regulares, $r_{ij}^{(k)}$ é sempre uma expressão regular, para todo $k \in \mathbb{N}$. Para $n = |S|$, a expressão regular $r_{ij}^{(n)}$ descreve a linguagem determinada pelos percursos generalizados de i para j que podem passar por qualquer estado do autómato. Assim, se $\sigma(F) = \{\text{números atribuídos aos estados finais}\}$, a expressão regular da linguagem que o autómato reconhece é dada por:

$$\sum_{f \in \sigma(F)} r_{1f}^{(n)}$$

Não seria necessário identificar o estado inicial por 1, mas esta expressão deveria ser alterada (cf. Exemplo 56).

Exemplo 55 Consideremos novamente o diagrama de transição do Exemplo 53.



Por aplicação do método de Kleene, vamos ver que a expressão regular $r_{11}^{(2)} + r_{12}^{(2)} = \mathtt{b}^\star + \mathtt{b}^\star\mathtt{a}\mathtt{c}^\star$ descreve a linguagem aceite pelo autómato.

$$r_{11}^{(0)} = \varepsilon + \mathtt{b} \qquad r_{22}^{(0)} = \varepsilon + \mathtt{c} \qquad r_{12}^{(0)} = \mathtt{a} \qquad r_{21}^{(0)} = \emptyset$$

$$r_{11}^{(1)} = r_{11}^{(0)} + r_{11}^{(0)}(r_{11}^{(0)})^\star r_{11}^{(0)} = \varepsilon + \mathtt{b} + (\varepsilon + \mathtt{b})(\varepsilon + \mathtt{b})^\star(\varepsilon + \mathtt{b}) = \mathtt{b}^\star$$

$$r_{22}^{(1)} = r_{22}^{(0)} + r_{21}^{0}(r_{11}^{(0)})^\star r_{12}^{(0)} = \varepsilon + \mathtt{c} + \emptyset(\varepsilon + \mathtt{b})^\star\mathtt{a} = \varepsilon + \mathtt{c}$$

$$r_{12}^{(1)} = r_{12}^{(0)} + r_{11}^{(0)}(r_{11}^{(0)})^\star r_{12}^{(0)} = \mathtt{a} + (\varepsilon + \mathtt{b})(\varepsilon + \mathtt{b})^\star\mathtt{a} = \mathtt{b}^\star\mathtt{a}$$

$$r_{21}^{(1)} = r_{21}^{(0)} + r_{21}^{(0)}(r_{11}^{(0)})^\star r_{11}^{(0)} = \emptyset$$

$$r_{11}^{(2)} = r_{11}^{(1)} + r_{12}^{(1)}(r_{22}^{(1)})^\star r_{21}^{(1)} = r_{11}^{(1)} = \mathtt{b}^\star$$

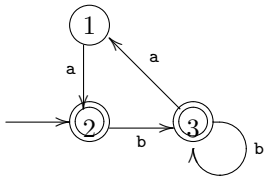
$$r_{12}^{(2)} = r_{12}^{(1)} + r_{12}^{(1)}(r_{22}^{(1)})^\star r_{22}^{(1)} = \mathtt{b}^\star\mathtt{a} + \mathtt{b}^\star\mathtt{a}(\varepsilon + \mathtt{c})^\star(\varepsilon + \mathtt{c}) = \mathtt{b}^\star\mathtt{a}\mathtt{c}^\star$$

$$r_{22}^{(2)} = r_{22}^{(1)} + r_{22}^{(1)}(r_{22}^{(1)})^\star r_{22}^{(1)} = \mathtt{c}^\star$$

$$r_{21}^{(2)} = \emptyset$$

Para evitar sobrecarregar a notação, podiamos ter escrito r_{ij}^k em vez de $r_{ij}^{(k)}$, mas causaria alguma confusão com a notação x^n introduzida antes. Se tivermos em conta o facto de o autómato anterior ser bastante simples, não é difícil concluir que este método se pode tornar (muito) lento.

Exemplo 56 Neste exemplo, restringimos o cálculo das expressões $r_{ij}^{(n)}$ às relevantes. A expressão que descreve a linguagem é $r_{22}^{(3)} + r_{23}^{(3)}$.



Como pretendemos $r_{22}^{(3)} + r_{23}^{(3)}$, só calculamos estas duas expressões para $k = 3$, e como $r_{22}^{(3)} = r_{22}^{(2)} + r_{23}^{(2)}(r_{33}^{(2)})^\star r_{32}^{(2)}$ e $r_{23}^{(3)} = r_{23}^{(2)} + r_{23}^{(2)}(r_{33}^{(2)})^\star r_{33}^{(2)}$, limitar-nos-emos a calcular as expressões relevantes também quando $k = 2$.

$k=0$	$k=1$	$k=2$
$r_{11}^{(0)} = \varepsilon$	$r_{11}^{(1)} = r_{11}^{(0)} + r_{11}^{(0)}(r_{11}^{(0)})^\star r_{11}^{(0)} \equiv \varepsilon$	
$r_{12}^{(0)} = \mathtt{a}$	$r_{12}^{(1)} = r_{12}^{(0)} + r_{11}^{(0)}(r_{11}^{(0)})^\star r_{12}^{(0)} \equiv r_{12}^{(0)} = \mathtt{a}$	
$r_{13}^{(0)} = \emptyset$	$r_{13}^{(1)} = r_{13}^{(0)} + r_{11}^{(0)}(r_{11}^{(0)})^\star r_{13}^{(0)} \equiv r_{13}^{(0)} = \emptyset$	
$r_{21}^{(0)} = \emptyset$	$r_{21}^{(1)} = r_{21}^{(0)} + r_{21}^{(0)}(r_{11}^{(0)})^\star r_{11}^{(0)} \equiv r_{21}^{(0)} = \emptyset$	
$r_{22}^{(0)} = \varepsilon$	$r_{22}^{(1)} = r_{22}^{(0)} + r_{21}^{(0)}(r_{11}^{(0)})^\star r_{12}^{(0)} = \varepsilon + \emptyset \equiv \varepsilon$	$r_{22}^{(2)} = r_{22}^{(1)} + r_{22}^{(1)}(r_{22}^{(1)})^\star r_{22}^{(1)} \equiv \varepsilon$
$r_{23}^{(0)} = \mathtt{b}$	$r_{23}^{(1)} = r_{23}^{(0)} + r_{21}^{(0)}(r_{11}^{(0)})^\star r_{13}^{(0)} = \mathtt{b} + \emptyset \equiv \mathtt{b}$	$r_{23}^{(2)} = r_{23}^{(1)} + r_{22}^{(1)}(r_{22}^{(1)})^\star r_{23}^{(1)} \equiv \mathtt{b}$
$r_{31}^{(0)} = \mathtt{a}$	$r_{31}^{(1)} = r_{31}^{(0)} + r_{31}^{(0)}(r_{11}^{(0)})^\star r_{11}^{(0)} \equiv \mathtt{a}$	
$r_{32}^{(0)} = \emptyset$	$r_{32}^{(1)} = r_{32}^{(0)} + r_{31}^{(0)}(r_{11}^{(0)})^\star r_{12}^{(0)} \equiv \emptyset + \mathtt{a}\varepsilon\mathtt{a} \equiv \mathtt{aa}$	$r_{32}^{(2)} = r_{32}^{(1)} + r_{32}^{(1)}(r_{22}^{(1)})^\star r_{22}^{(1)} \equiv \mathtt{aa}$
$r_{33}^{(0)} = \varepsilon + \mathtt{b}$	$r_{33}^{(1)} = r_{33}^{(0)} + r_{31}^{(0)}(r_{11}^{(0)})^\star r_{13}^{(0)} \equiv \varepsilon + \mathtt{b} + \emptyset \equiv \varepsilon + \mathtt{b}$	$r_{33}^{(2)} = r_{33}^{(1)} + r_{32}^{(1)}(r_{22}^{(1)})^\star r_{23}^{(1)} \equiv \varepsilon + \mathtt{b} + \mathtt{aab}$

Finalmente, temos

$$r_{22}^{(3)} = r_{22}^{(2)} + r_{23}^{(2)}(r_{33}^{(2)})^\star r_{32}^{(2)} = \varepsilon + \mathtt{b}(\varepsilon + \mathtt{b} + \mathtt{aab})^\star\mathtt{aa} \equiv \varepsilon + \mathtt{b}(\mathtt{b} + \mathtt{aab})^\star\mathtt{aa}$$

$$r_{23}^{(3)} = r_{23}^{(2)} + r_{23}^{(2)}(r_{33}^{(2)})^\star r_{33}^{(2)} \equiv \mathtt{b}(\varepsilon + \mathtt{b} + \mathtt{aab})^\star \equiv \mathtt{b}(\mathtt{b} + \mathtt{aab})^\star$$

obtendo a expressão procurada

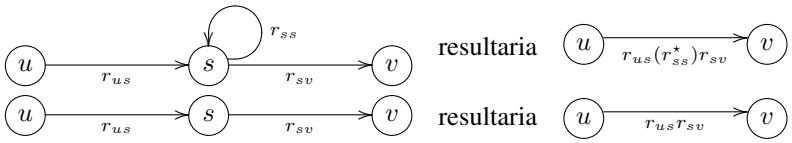
$$r_{22}^{(3)} + r_{23}^{(3)} \equiv \varepsilon + \mathtt{b}(\mathtt{b} + \mathtt{aab})^\star\mathtt{aa} + \mathtt{b}(\mathtt{b} + \mathtt{aab})^\star \equiv \varepsilon + \mathtt{b}(\mathtt{b} + \mathtt{aab})^\star(\mathtt{aa} + \varepsilon)$$

3.3.2 Método de eliminação de estados (Brzozowski-McCluskey)

Para obter uma expressão regular que defina a linguagem que um dado autómato finito aceita, é mais simples aplicar o método de "eliminação de estados". Neste método, etiquetamos os arcos com expressões regulares que representam o conjunto de palavras que permitiriam essa transição. À medida que os estados vão sendo eliminados, são acrescentados novos arcos (transições) ou mais informação às expressões correspondentes, se esses arcos já existiam.

O **método de eliminação de estados** consiste no seguinte.

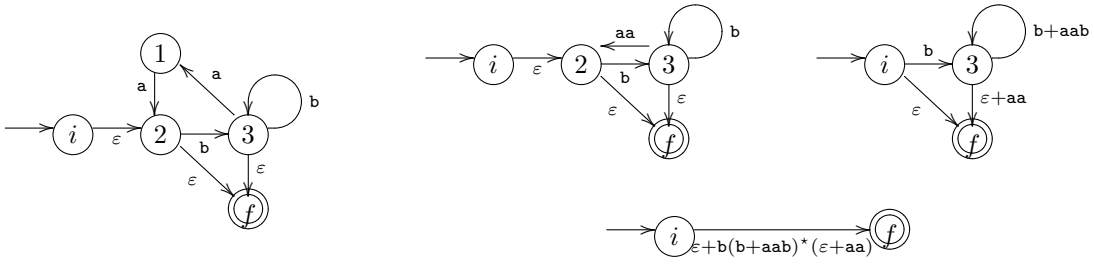
- Remover todos os estados que não são acessíveis do estado inicial do autómato ou que não permitem aceder a algum estado final do autómato.
- Se não sobrar nenhum estado final, dar como resultado $\emptyset$ e terminar.
- Se o estado inicial s_0 tiver grau de entrada não nulo, introduzir um estado inicial i com transição-ε para s_0.
- Se existirem vários estados finais ou apenas um mas com grau de saída não nulo, introduzir um novo estado final f que passará a ser o único estado final, acrescentando transições-ε dos que antes eram finais para f.
- Substituir as etiquetas dos arcos do diagrama por expressões regulares.
- Eliminar os estados um a um, com excepção do inicial e do final. Para eliminar um estado s, substituir cada par de ramos (u, s) e (s, v), com $u \neq s$ e $v \neq s$, por um novo ramo (u, v), cuja etiqueta é obtida assim:



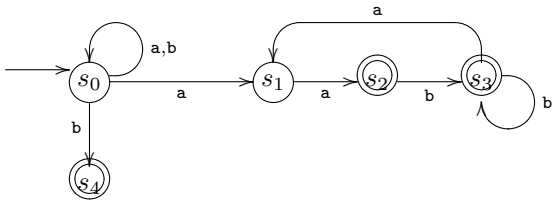
Se o ramo (u, v) já existir anteriormente, sendo r_{uv} a expressão associada, então substituir essa expressão por $r_{uv} + r_{us}(r_{ss}^{\star})r_{sv}$ ou $r_{uv} + r_{us}r_{sv}$, respetivamente.

Os estados podem ser eliminados por qualquer ordem. Contudo, a complexidade da expressão final e das expressões obtidas em passos intermédios pode depender da ordem pela qual os estados são eliminados.

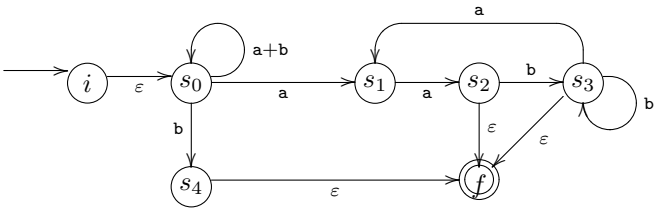
Exemplo 57 Por aplicação do método de eliminação de estados, concluimos que $\varepsilon + \mathtt{b}(\mathtt{b} + \mathtt{aab})^{\star}(\varepsilon + \mathtt{aa})$ descreve a linguagem reconhecida pelo autómato considerado no Exemplo 56:



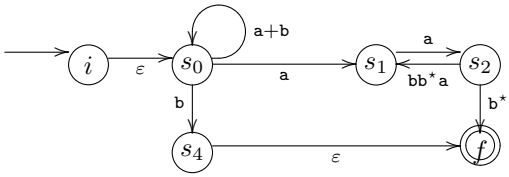
Exemplo 58 Vamos aplicar o método de eliminação de estados para determinar uma expressão regular que descreva a linguagem aceite pelo autómato finito representado pelo diagrama seguinte.



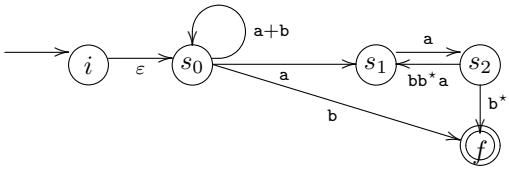
Começamos por introduzir os estados i e f referidos e por substituir todas as etiquetas por expressões regulares. Para isso, substituimos o lacete múltiplo em s_0 por um único etiquetado com a expressão $\mathtt{a}+\mathtt{b}$. O novo diagrama está representado abaixo. Já não se trata de um autómato pois as etiquetas são expressões regulares.



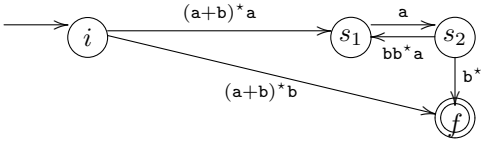
Na eliminação de s_3, é necessário substituir a expressão de (s_2, f) por $\varepsilon + \mathtt{b}\mathtt{b}^\star\varepsilon \equiv \mathtt{b}^\star$, e criar um arco (s_2, s_1) com etiqueta $\mathtt{b}\mathtt{b}^\star\mathtt{a}$:



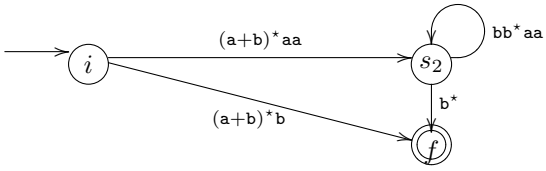
Para eliminar s_4, substitui-se $s_0 \to s_4 \to f$ pelo arco (s_0, f):



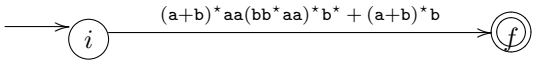
Para eliminar s_0, substitui-se $i \to s_0 \to f$ e $i \to s_0 \to s_1$, criando os arcos (i, f) e (i, s_1):



Para eliminar s_1, há que considerar $s_2 \to s_1 \to s_2$ e $i \to s_1 \to s_2$, pelo que os pares afetados são (s_2, s_2) e (i, s_2):



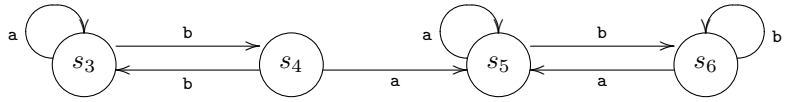
Finalmente, elimina-se s_2:



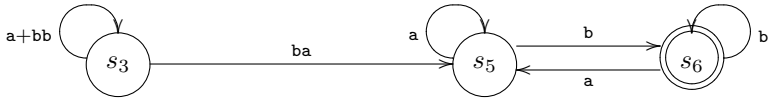
A linguagem aceite pelo autómato é descrita pela expressão regular $(\mathtt{a}+\mathtt{b})^\star\mathtt{aa}(\mathtt{bb}^\star\mathtt{aa})^\star\mathtt{b}^\star + (\mathtt{a}+\mathtt{b})^\star\mathtt{b}$.

NB: os diagramas acima apresentados não representam autómatos finitos!

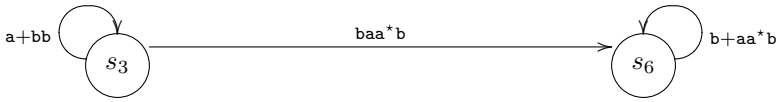
Exemplo 59 Consideremos o diagrama seguinte (excerto de um autómato). Usando eliminação de estados, vamos simplificá-lo de modo a obter uma expressão regular que descreva as linguagem das palavras que levam o autómato do estado s_3 ao estado s_6.



Para eliminar s_4, devemos: substituir o percurso $s_3s_4s_3$ por um lacete em s_3 etiquetado por bb (acrescentando essa informação ao lacete já existente); substituir o percurso $s_3s_4s_5$ por um arco de s_3 para s_5 etiquetado por ba. Obtém-se o diagrama seguinte:

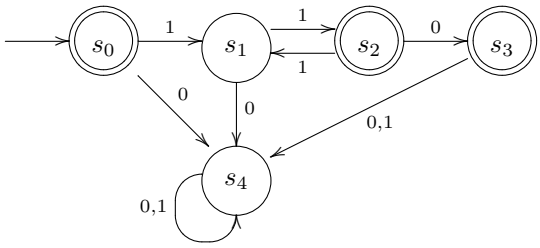


Agora, por eliminação de s_5, obtem-se:

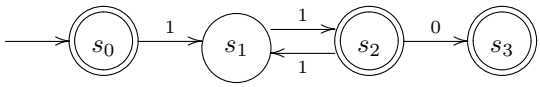


A expressão $(\mathtt{a}+\mathtt{bb})^\star\mathtt{baa}^\star\mathtt{b}(\mathtt{b}+\mathtt{aa}^\star\mathtt{b})^\star$ descreve a linguagem das palavras que levam o autómato de s_3 a s_6. Notemos que não aplicámos o método de eliminação, mas apenas efetuámos uma análise com base na técnica subjacente.

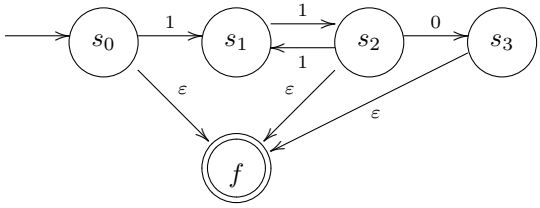
Exemplo 60 Pretendemos obter, pelo método de eliminação de estados, uma expressão regular que descreva a linguagem reconhecida pelo autómato seguinte.



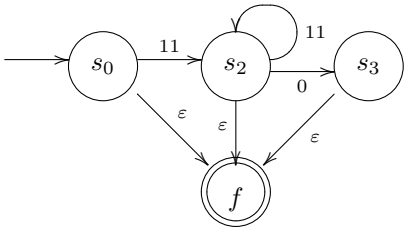
Eliminamos s_4, pois não permite aceder a estados finais.



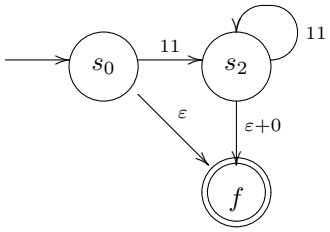
Acrescentamos um novo estado final que passará a ser o único final.



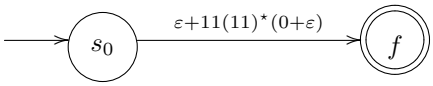
Todas as etiquetas podem ser entendidas como expressões regulares. Passamos à eliminação de s_1.



A seguir, eliminamos s_3:



E finalmente, eliminamos s_2:



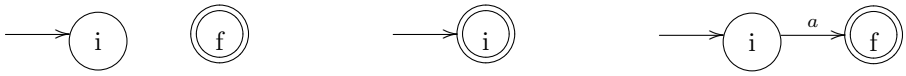
Logo, a expressão $\varepsilon + \texttt{11}(\texttt{11})^\star(\texttt{0} + \varepsilon)$ descreve a linguagem que o autómato dado reconhece, a qual é equivalente a $\varepsilon + \texttt{11}(\texttt{11})^\star + \texttt{11}(\texttt{11})^\star\texttt{0}$, ou simplesmente, $(\texttt{11})^\star + \texttt{11}(\texttt{11})^\star\texttt{0}$.

3.3.3 Método de McNaughton-Yamada-Thompson

Resta apresentar o método de McNaughton-Yamada-Thompson, referido na prova da Proposição 16. Este método constrói um AFND-ε com **um único estado final, do qual não há transições,** que aceita a linguagem descrita pela expressão regular dada. Segue uma abordagem recursiva que pode ser traduzida como uma prova (construtiva) por indução sobre *o número de operadores na expressão regular*. Por isso, apresentamo-lo dessa forma.

Seja r a expressão regular de alfabeto Σ dada. Se r tem 0 operadores, então $r = \emptyset$, ou $r = \varepsilon$ ou $r = a$, para algum $a \in \Sigma$, e as linguagens descritas são aceites pelos autómatos seguintes, os quais estão nas condições impostas.

$r = \emptyset$ $\qquad$ $r = \varepsilon$ $\qquad$ $r = a$



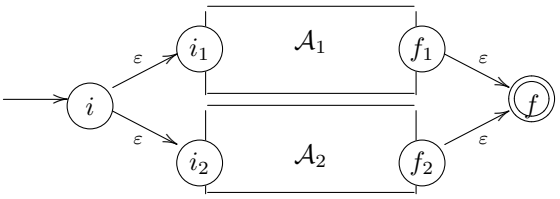
Como hipótese de indução, supomos que, para qualquer expressão regular com k operadores, sendo $0 \leq k \leq n$ e $n \geq 0$ fixo mas arbitrário, é possível descrever um AFND-ε que aceita a linguagem por ela descrita e tem um único estado final, não havendo transições desse estado. E, vamos mostrar que, então, dada uma qualquer expressão regular r com $n+1$ operadores, ainda é possível descrever um tal autómato.

Se r tem $n+1$ operadores, então ou $r = (r_1 + r_2)$, ou $r = (r_1 r_2)$ ou $r = (r_1^\star)$. O número de operadores em r_1 e em r_2 não excede n, pelo que, por hipótese de indução, existem autómatos finitos que aceitam as linguagens descritas por essas subexpressões e que estão nas condições enunciadas (um só estado final, não havendo transições desse estado).

Se for $r = (r_1 + r_2)$, sejam $\mathcal{A}_1$ e $\mathcal{A}_2$ os autómatos que aceitam $\mathcal{L}(r_1)$ e $\mathcal{L}(r_2)$, respetivamente (os quais, por hipótese, foram já construídos pelo mesmo método). Como $\mathcal{L}(r) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$, a ideia vai ser construir a partir de $\mathcal{A}_1$ e $\mathcal{A}_2$ um autómato que aceite $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. Sejam $\mathcal{A}_1 = (S_1, \Sigma, \delta_1, i_1, \{f_1\})$ e $\mathcal{A}_2 = (S_2, \Sigma, \delta_2, i_2, \{f_2\})$ tais autómatos. Sejam i e f dois novos estados, e defina-se

$$\mathcal{A}_\cup = (S_1 \cup S_2 \cup \{i, f\}, \Sigma, \delta, i, \{f\})$$

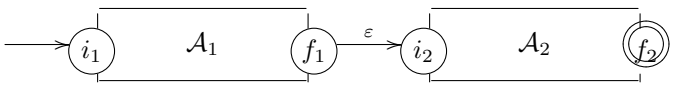
em que $\delta = \delta_1 \cup \delta_2 \cup \{(i, \varepsilon, i_1), (i, \varepsilon, i_2), (f_1, \varepsilon, f), (f_2, \varepsilon, f)\}$. Podemos mostrar que $\mathcal{A}_\cup$ reconhece $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$, e está nas condições requeridas.



De modo análogo, se for $r = (r_1 r_2)$, então sejam $\mathcal{A}_1$ e $\mathcal{A}_2$ os autómatos que aceitam $\mathcal{L}(r_1)$ e $\mathcal{L}(r_2)$ respetivamente (os quais, por hipótese, foram já construídos pelo mesmo método). Podemos mostrar que o autómato

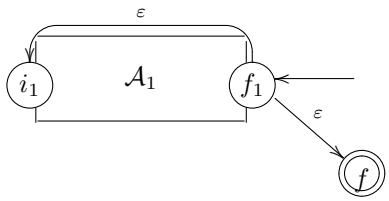
$$\mathcal{A}_\bullet = (S_1 \cup S_2, \Sigma, \delta_1 \cup \delta_2 \cup \{(f_1, \varepsilon, i_2)\}, i_1, \{f_2\})$$

reconhece $\mathcal{L}(\mathcal{A}_1)\mathcal{L}(\mathcal{A}_2)$, e está nas condições pretendidas.



E finalmente, se for $r = (r_1^\star)$, seja $\mathcal{A}_1 = (S_1, \Sigma, \delta_1, i_1, \{f_1\})$ (o qual, por hipótese, foi já construído pelo mesmo método) e seja f um novo estado. Constrói-se o autómato seguinte, podendo-se mostrar que aceita $\mathcal{L}(\mathcal{A}_1)^\star$.

$$\mathcal{A}_\star = (S_1 \cup \{f\}, \Sigma, \delta_1 \cup \{(f_1, \varepsilon, i_1), (f_1, \varepsilon, f)\}, f_1, \{f\})$$

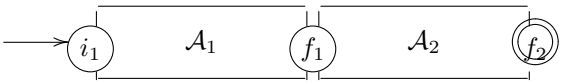


Em suma:

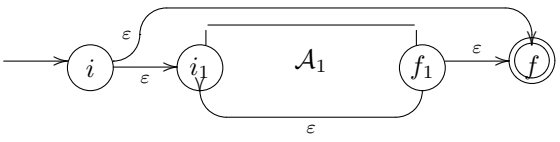
- mostrámos que qualquer linguagem definida por uma expressão regular com zero operadores pode ser reconhecida por um autómato finito (AFND-ε com um único estado final, do qual não saem transições).
- mostrámos ainda que se supusermos que as linguagens definidas por expressões regulares com até n operadores podem ser reconhecidas por autómatos finitos (AFNDs-ε com um único estado final, do qual não saem transições), então também as linguagens definidas por expressões regulares com $n + 1$ operadores o podem ser, sendo $n \geq 0$ **fixo** mas qualquer.

Logo, pelo princípio de indução, conclui-se que qualquer que seja o número de operadores da expressão regular dada, existe um autómato que aceita a linguagem que a expressão descreve.

Construções alternativas. Sem perda de correção, a construção para $r = (r_1 r_2)$ pode ser simplificada, pois podemos simplesmente identificar o estado final $\mathcal{A}_1$ com o estado inicial de $\mathcal{A}_2$. Ainda que a junção dos diagramas seja simples, tal obrigaria a algum cuidado extra na definição de δ em $\mathcal{A}_\bullet$ para que apenas f_1 (ou i_2) fosse referido,

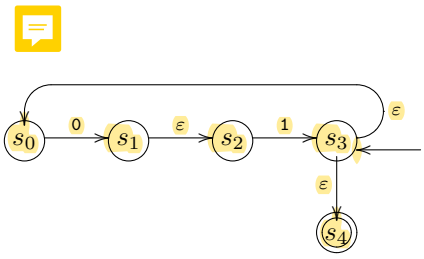


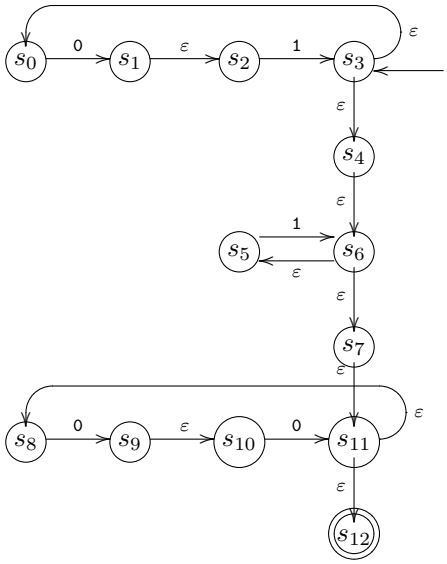
Também para $(r_1^\star)$, podiamos considerar $\mathcal{A}_\star = (S_1 \cup \{i, f\}, \Sigma, \delta_1 \cup \{(i, \varepsilon, i_1), (f_1, \varepsilon, i_1), (i, \varepsilon, f)\}, i, \{f\})$.



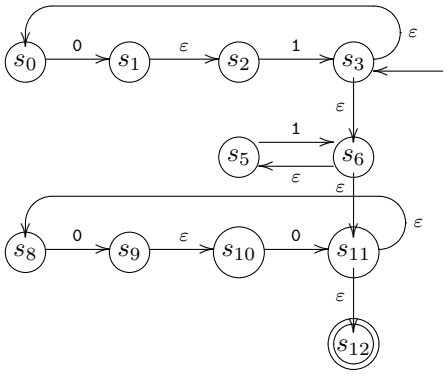
Provas construtivas. Observemos novamente o facto de as provas dos principais resultados que estudaremos serem construtivas. Para aplicar os métodos que apresentam para resolver problemas (de conversão, por exemplo) ou para implementar esses métodos numa linguagem de programação é necessário conhecê-las. Na resolução de problemas "à-mão", nem sempre é menos trabalhoso seguir o método sugerido na prova, como se ilustra no Exemplo 61.

Exemplo 61 Seja $L = \mathcal{L}(\ (01)^\star 1^\star (00)^\star\ )$. Esta linguagem pode ser vista como $L_1 L_2$ com $L_1 = \mathcal{L}(\ (01)^\star\ )$ e $L_2 = \mathcal{L}(\ 1^\star (00)^\star\ )$. O AFND-$\varepsilon$ construído para L_1 (na versão original) está à esquerda e para $L = \mathcal{L}(\ (01)^\star 1^\star (00)^\star\ )$, à direita:





Claramente, este autómato é equivalente (no sentido de ambos reconhecerem a mesma linguagem) ao representado pelo diagrama de transição seguinte, o qual é ligeiramente mais simples.

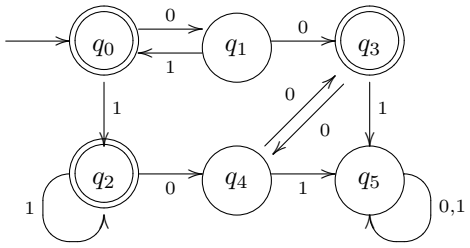


Se tivessemos usado a construção alternativa para a concatenação, a transição-ε de s_1 para s_2 não teria sido criada pois os estados s_2 e s_1 seriam aglutinados num só, bem como s_9 e s_{10}. Mas, não vamos simplificar mais o autómato.

Seguindo a ideia da prova da Proposição 14, vamos determinar um autómato finito determinístico equivalente ao último representado. O estado inicial desse AFD é dado por $\text{Fecho}_\varepsilon(s_3)$, sendo o conjunto dos estados acessíveis do estado inicial do AFND-ε, e é $\{s_3, s_0, s_6, s_5, s_{11}, s_8, s_{12}\}$. A função de transição do AFD é apresentada na tabela seguinte, onde a coluna mais à esquerda indica o estado em que o AFD está.

Estado	0	1
$\{s_0, s_3, s_5, s_6, s_8, s_{11}, s_{12}\}$	$\{s_1, s_2, s_9, s_{10}\}$	$\{s_5, s_6, s_8, s_{11}, s_{12}\}$
$\{s_1, s_2, s_9, s_{10}\}$	$\{s_8, s_{11}, s_{12}\}$	$\{s_0, s_3, s_5, s_6, s_8, s_{11}, s_{12}\}$
$\{s_5, s_6, s_8, s_{11}, s_{12}\}$	$\{s_9, s_{10}\}$	$\{s_5, s_6, s_8, s_{11}, s_{12}\}$
$\{s_8, s_{11}, s_{12}\}$	$\{s_9, s_{10}\}$	$\{\,\}$
$\{s_9, s_{10}\}$	$\{s_8, s_{11}, s_{12}\}$	$\{\,\}$
$\{\,\}$	$\{\,\}$	$\{\,\}$

Se denotarmos os estados $\{s_0, s_3, s_5, s_6, s_8, s_{11}, s_{12}\}$, $\{s_1, s_2, s_9, s_{10}\}$, $\{s_5, s_6, s_8, s_{11}, s_{12}\}$, $\{s_8, s_{11}, s_{12}\}$, $\{s_9, s_{10}\}$, e $\{\,\}$ por q_0, q_1, q_2, q_3, q_4 e q_5, respetivamente, o AFD obtido é:



Por mero acaso, tal autómato é o AFD mínimo (isto é, com o menor número de estados) que aceita $\mathcal{L}((\texttt{01})^\star\texttt{1}^\star(\texttt{00})^\star)$. Podemos tentar justificar esse facto, analisando o que cada estado desse autómato está a memorizar sobre palavra que é lida. No próximo capítulo, estudaremos o teorema de Myhill-Nerode, que ajuda a sistematizar a justificação.

Sumariando. Dos resultados demonstrados neste capítulo concluimos que, dada uma linguagem $L \subseteq \Sigma^\star$, é equivalente dizer que "L é regular", ou dizer "L é aceite por um AFD", ou dizer "L é aceite por um AFND", ou ainda "L é aceite por um AFND-ε".

Capítulo 4

Propriedades das linguagens regulares

Imaginando que *classificamos* as linguagens de alfabeto Σ segundo algum critério, podemos referir-nos ao conjunto das linguagens regulares como a **classe das linguagens regulares**. Sabemos já que é também a classe das linguagens que são reconhecidas por autómatos finitos. Existem linguagens que não são regulares? A resposta é afirmativa. Vamos estudar alguns exemplos e resultados fundamentais sobre a classe de linguagens regulares, nomeadamente:

- A classe das linguagens regulares é fechada para a união finita, a complementação, a intersecção, a concatenação (finita) e o fecho de Kleene.

- Uma linguagem L é regular **se e só se** o conjunto de classes de equivalência da relação $\mathcal{R}_L$ é finito, sendo $\mathcal{R}_L$ a relação de equivalência em $\Sigma^\star$ definida por $x\mathcal{R}_L y$ se e só se $\forall z \in \Sigma^\star \ (xz \in L \Leftrightarrow yz \in L)$, com $x, y \in \Sigma^\star$.

- O AFD com menor número de estados que aceita uma dada linguagem regular L é único (a menos da designação dos estados). O conjunto de estados do **AFD mínimo** que aceita L corresponde ao conjunto das classes de equivalência de $\mathcal{R}_L$ (e também a função de transição δ é determinada por essa relação).

- **Se** L é uma linguagem regular **então** existe uma constante $n \in \mathbb{N}$ tal que, qualquer que seja $x \in L$, se $|x| \geq n$ então existem $u, v, w \in \Sigma^\star$ tais que $x = uvw$, $|uv| \leq n$ e $|v| \geq 1$, e $\forall i \geq 0 \ \ uv^i w \in L$. Mais ainda, n não excede o número de estados do menor autómato finito que aceita L.
  (Existem linguagens que a satisfazem esta condição e não são regulares, o que significa que a condição é necessária mas não suficiente para que uma linguagem seja regular).

Nas próximas secções, apresentamos as provas destes resultados.

4.1 Propriedades de fecho

Diz-se que um conjunto é **fechado para uma operação** binária Θ se e só se quaisquer que sejam os elementos x e y desse conjunto, $x\ \Theta\ y$ ainda é um elemento do conjunto. Se a operação Θ é unária, então o conjunto é fechado para a operação se e só se para todo x no conjunto, $\Theta\ x$ ainda é um elemento do conjunto.

Exemplo 62 O conjunto dos números reais não negativos, é fechado para a soma mas não é fechado para a subtracção, pois $\forall x \in \mathbb{R}_0^+\ \forall y \in \mathbb{R}_0^+\ \ x+y \in \mathbb{R}_0$, mas como $3-5 \notin \mathbb{R}_0$, vemos que $\exists x \in \mathbb{R}_0^+\ \exists y \in \mathbb{R}_0^+\ \ x-y \notin \mathbb{R}_0$, o que quer dizer que não é verdade que $\forall x \in \mathbb{R}_0^+\ \forall y \in \mathbb{R}_0^+\ \ x-y \in \mathbb{R}_0$.

As linguagens regulares sobre Σ constituem o menor conjunto de linguagens sobre Σ que é fechado para a união (finita), concatenação e fecho de Kleene. Na Proposição 17, consideramos também outras propriedades.

Proposição 17 *A classe de linguagens regulares é fechada para a união finita, a complementação, a intersecção, a concatenação (finita) e o fecho de Kleene.*

Prova:

- O fecho de Kleene de uma linguagem regular é uma linguagem regular.

  Se L é uma linguagem regular então, por definição, L é descrita por alguma expressão regular. Seja r tal expressão. Sabemos que $L^\star$, a que se chama o fecho de Kleene de L, é descrita por $r^\star$, se r descrever L.

- A concatenação de duas linguagens regulares é uma linguagem regular.

  Se L_1 e L_2 são regulares, existem expressões regulares r_1 e r_2 que as descrevem. Então, $r_1 r_2$ descreve $L_1 L_2$. Portanto, $L_1 L_2$ é regular. A prova de que $L_1 L_2 \cdots L_n$ é regular se cada uma das linguagens L_i o for, para $1 \leq i \leq n$, pode ser feita por indução, como a seguinte.

- A união finita de linguagens regulares é regular, ou seja, $\bigcup_{i=1}^{n} L_i$ é regular, quaisquer que sejam $n \geq 2$ e as linguagens regulares $L_i \subseteq \Sigma^\star$, com $2 \leq i \leq n$.

  Prova por indução sobre n: Se L_1 e L_2 são regulares, existem expressões regulares r_1 e r_2 que as descrevem. Então, $r_1 + r_2$ descreve $L_1 \cup L_2$ pelo que $L_1 \cup L_2$ é regular.

  Dado $n \geq 2$, suponhamos, como hipótese de indução, que quaisquer que sejam $L_1, \ldots, L_n$ linguagens regulares de alfabeto Σ, a união $L_1 \cup \ldots \cup L_n$ é regular. E, vamos mostrar que se $Q_1, \ldots Q_n, Q_{n+1}$ forem $n+1$ linguagens regulares de alfabeto Σ então $Q_1 \cup \ldots \cup Q_n \cup Q_{n+1}$ é regular.

  De facto, $Q_1 \cup \ldots \cup Q_n \cup Q_{n+1} = (Q_1 \cup \ldots \cup Q_n) \cup Q_{n+1}$ e por hipótese de indução, $Q_1 \cup \ldots \cup Q_n$ é regular, pelo que concluimos que $(Q_1 \cup \ldots \cup Q_n) \cup Q_{n+1}$ é união de duas linguagens regulares. Como mostrámos que a união de *duas* linguagens regulares é regular, concluimos que $Q_1 \cup \ldots \cup Q_n \cup Q_{n+1}$ é regular.

- O complementar de uma linguagem regular é regular, ou seja, se $L \subseteq \Sigma^\star$ é regular então $\Sigma^\star \backslash L$ é regular.

  Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD tal que $L = \mathcal{L}(\mathcal{A})$. Sendo δ, por definição, uma função total de $S \times \Sigma$ em S, o autómato $\mathcal{A}' = (S, \Sigma, \delta, s_0, S \backslash F)$ é um AFD que aceita $\Sigma^\star \backslash L$, pelo que $\Sigma^\star \backslash L$ é regular.

- A intersecção de duas linguagens regulares é regular.

  Como $\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2$, podemos usar os dois resultados anteriores para concluir que $L_1 \cap L_2$ é regular se L_1 e L_2 forem regulares. □

Exemplo 63 Sejam L_1, L_2, e L_3 as linguagens de alfabeto $\{\mathtt{0}, \mathtt{a}, \mathtt{b}\}$ assim definidas:

$$\begin{aligned} L_1 &= \{\mathtt{a}\} \cup \{\mathtt{b}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^\star\} \\ L_2 &= \{w \in \{\mathtt{a}, \mathtt{b}\}^\star \mid \ |w| = 2 \ \vee \ |w| = 3\} \\ L_3 &= L_1(\{\mathtt{0}\}L_2)^\star\{\mathtt{0}\} \end{aligned}$$

As linguagens L_1, L_2 e L_3 são regulares: são definidas pelas expressões regulares $\mathtt{a}+\mathtt{b}(\mathtt{a}+\mathtt{b})^\star$, $(\mathtt{a}+\mathtt{b})(\mathtt{a}+\mathtt{b})(\varepsilon+\mathtt{a}+\mathtt{b})$ e $(\mathtt{a}+\mathtt{b}(\mathtt{a}+\mathtt{b})^\star)(\mathtt{0}(\mathtt{a}+\mathtt{b})(\mathtt{a}+\mathtt{b})(\varepsilon+\mathtt{a}+\mathtt{b}))^\star\mathtt{0}$, respetivamente. Pela Proposição 17, concluimos também que

$$\overline{L_3} \cap (\ \{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\}\{\mathtt{0}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^\star\}\ )$$

é uma linguagem regular, considerando o seguinte.

- Como L_3 é regular, também $\overline{L_3}$.

- $\{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\}$ é regular porque é união finita de linguagens regulares, pois

  $$\{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\} = \bigcup_{n=0}^{1000} \{\mathtt{b}^n\mathtt{a}^n\}.$$

  Se designarmos por r_n a expressão regular que descreve $\{\mathtt{b}^n\mathtt{a}^n\}$, podemos defini-la por recorrência do modo seguinte: $r_0 = \varepsilon$ e $r_{k+1} = \mathtt{a}r_k\mathtt{b}$, para $k \geq 0$. Salientemos que r_n descreve uma linguagem que é constituída apenas por uma palavra — a palavra que tem n $\mathtt{a}$'s e n $\mathtt{b}$'s, e que não tem $\mathtt{a}$'s à direita de $\mathtt{b}$'s.

- $\{\mathtt{0}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^\star\}\ )$ é regular, pois é definida pela expressão $\mathtt{0}(\mathtt{a}+\mathtt{b})^\star$.

Logo, $\{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\}\{\mathtt{0}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^\star\}$ é regular, já que é concatenação de linguagens regulares. E, portanto, $\overline{L_3} \cap (\ \{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\}\{\mathtt{0}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^\star\}\ )$ é regular porque é intersecção de duas linguagens regulares.

Exercício 4.1.1 Justificar que qualquer que seja a linguagem L, se L é finita (isto é, se L é um conjunto palavras que é finito) então L é regular. Concluir que $\{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\}$ é regular pois é finita.

Exercício 4.1.2 Sejam L_1 e L_2 linguagens de alfabeto Σ tais que $L_1 \cap L_2 = \emptyset$, e L_2 é regular.

a) Mostrar que se $L_1 \cup L_2$ é regular então L_1 é regular.

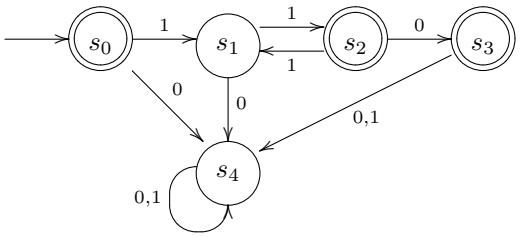
b) Mostrar que $L_2 \cup L_1$ não é regular se e só se L_1 não é regular.

No Exercício 4.1.2, a condição $L_1 \cap L_2 = \emptyset$ é fundamental. De facto, se $L_1 \cap L_2 \neq \{\ \}$, podemos facilmente encontrar linguagens L_1 e L_2 tais que L_2 é regular, L_1 não é regular e $L_1 \cup L_2$ é regular. Basta tomar $L_2 = \Sigma^\star$ e $L_1 = \{a^n \mid n \text{ primo}\}$, sendo a algum elemento de Σ. Na secção 4.3 mostraremos que L_1 não é regular.

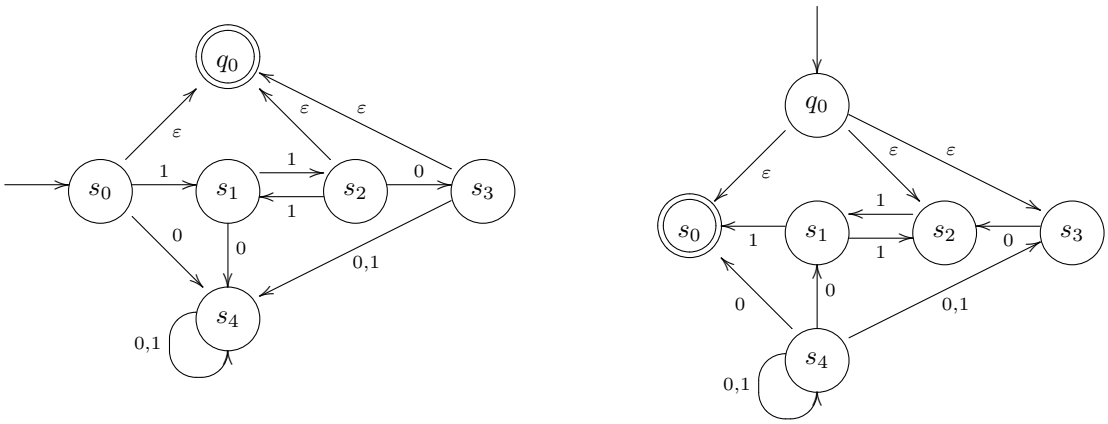
Fecho para o reverso

Para $x \in \Sigma^\star$, denotamos por x^r a palavra x escrita da direita para a esquerda. Formalmente, $\varepsilon^r = \varepsilon$ e $(aw)^r = w^r a$, para todo $w \in \Sigma^\star$ e $a \in \Sigma$. Dada uma linguagem L, seja $L^r = \{x^r \mid x \in L\}$.

Exemplo 64 A prova da Proposição 18 descreve um método de conversão de um autómato finito que reconhece L num autómato finito que que reconhece L^r. A ideia é proceder à inversão do sentido das transições após a criação de um único estado final, como ilustramos para o autómato representado abaixo.



O autómato construído para L^r pelo método de conversão encontra-se à direita. À esquerda, vemos um autómato equivalente ao inicial e que pode ser entendido como um passo intermédio na transformação.



Proposição 18 *Qualquer que seja a linguagem L, tem-se L é regular se e só se L^r é regular.*

Prova: Se mostrarmos que *se L é regular então L^r é regular*, concluimos que se L^r é regular então $(L^r)^r$ é regular. E, como $(L^r)^r = L$, ficará provado que se L^r é regular então L é regular.

Assim, vamos mostrar que se L é regular então L^r é regular. Sabemos que se L é regular então existe um autómato finito determinístico que aceita L. Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um tal autómato.

Seja q_0 um novo estado. Consideremos um autómato finito $\mathcal{A}' = (S \cup \{q_0\}, \Sigma, \delta', q_0, \{s_0\})$ em que a função de transição δ' é dada por $\delta'(q_0, \varepsilon) = \{f \mid f \in F\}$ (transições do novo estado inicial para cada um dos estados finais de $\mathcal{A}$ por ε), e para os restantes estados $s \in S$ e símbolos $a \in \Sigma$, tem-se $\delta'(s', a) = \{s\}$ sse $\delta(s, a) = s'$ (ou seja, orientam-se os arcos em sentido inverso).

Não é difícil concluir que, existe em $\mathcal{A}$ um percurso γ etiquetado por x de s_0 para um estado final f se e só se existe em $\mathcal{A}'$ um percurso etiquetado por x^r do estado q_0 para o estado s_0, sendo esse percurso $(q_0, f)\gamma^r$, constituído pela transição por ε de q_0 para o estado f, seguida das efetuadas no processamento de x por $\mathcal{A}$, por ordem inversa. □

4.2 Autómato produto e aplicações

Dados dois autómatos finitos $\mathcal{A}_1$ e $\mathcal{A}_2$, podemos definir facilmente um AFND-ε que reconhece $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$, seguindo a ideia da construção descrita no método de McNaughton-Yamada-Thompson. Se pretendermos um AFD que reconheça $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$, podemos aplicar o método de conversão, embora possa ser trabalhoso (pois tem complexidade exponencial, no pior caso). Também a igualdade

$$\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2,$$

sugere um método para, com "algum" esforço, determinar o autómato que aceita a $L_1 \cap L_2$ a partir de dois autómatos finitos $\mathcal{A}_1$ e $\mathcal{A}_2$ tais que $L_1 = \mathcal{L}(\mathcal{A}_1)$ e $L_2 = \mathcal{L}(\mathcal{A}_2)$.

> *Obter AFDs $\mathcal{A}'_1$ e $\mathcal{A}'_2$ equivalentes a $\mathcal{A}_1$ e a $\mathcal{A}_2$. Para obter AFD para $\overline{L_1}$, considerar que os estados finais de $\mathcal{A}'_1$ passam a ser não-finais e os não-finais passam a finais. Proceder do mesmo modo para obter um AFD para $\overline{L_2}$. Por combinação destes dois, determinar um autómato finito que aceita $\overline{L_1} \cup \overline{L_2}$, convertê-lo para um AFD, e trocar estados finais por não finais para obter um autómato para $\overline{\overline{L_1} \cup \overline{L_2}}$.*

Existe um método muito mais simples e eficiente para obter um AFD que reconhece $L_1 \cap L_2$ se forem dados os AFDs para L_1 e L_2. Baseia-se na construção do *autómato produto*. Essa construção é útil também para obter um AFD para $L_1 \cup L_2$ e para decidir se $L_1 \subset L_2$, ou se $L_1 \neq L_2$ e ainda se $L_1 \cap L_2 = \emptyset$, ou mesmo se $L_1 = L_2$.

Autómato produto. Sejam $\mathcal{A}_1 = (S_1, \Sigma, \delta_1, i_1, F_1)$ e $\mathcal{A}_2 = (S_2, \Sigma, \delta_2, i_2, F_2)$ autómatos finitos determinísticos que reconhecem L_1 e L_2, respetivamente. Qualquer que seja $x \in \Sigma^\star$, tem-se:

- $x \in L_1 \cap L_2$ se e só se x leva ambos os autómatos a estado final;
- $x \in L_1 \cup L_2$ se e só se x leva pelo menos um dos autómatos a estado final.

O autómato produto vai, por assim dizer, *imitar simultaneamente* $\mathcal{A}_1$ *e* $\mathcal{A}_2$. Os seus estados são pares (s_1, s_2), com $s_1 \in S_1$ e $s_2 \in S_2$, sendo $S_1 \times S_2$ o seu conjunto de estados, e a função de transição δ é dada por

$$\delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a))$$

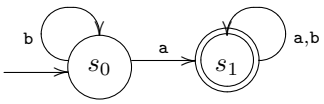
com $a \in \Sigma$ e $(s_1, s_2) \in S_1 \times S_2$ quaisquer. O seu estado inicial é (i_1, i_2). Podemos definir o conjunto de estados finais de forma a obter um AFD que aceita $L_1 \cap L_2$ ou que aceita $L_1 \cup L_2$. Assim, o conjunto de estados finais será

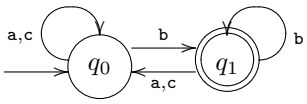
- $F_\cap = \{(s_1, s_2) \mid s_1 \in F_1 \ \wedge \ s_2 \in F_2\}$, para reconhecer $L_1 \cap L_2$;
- $F_\cup = \{(s_1, s_2) \mid s_1 \in F_1 \ \vee \ s_2 \in F_2\}$, para reconhecer $L_1 \cup L_2$.

Importa salientar a importância desta construção do ponto de vista computacional. O número de estados do autómato produto, $|S_1 \times S_2| = |S_1||S_2|$, é polinomial no número de estados dos AFDs de partida, no pior caso. Assim, conduz a algoritmos eficientes para problemas de decisão como os que mencionamos no exercício seguinte.

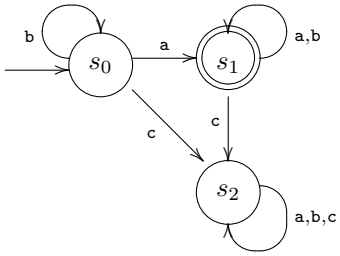
Exercício 4.2.1 Dados dois AFDs $\mathcal{A}_1$ e $\mathcal{A}_2$, como usar o autómato produto para determinar se $\mathcal{L}(\mathcal{A}_1) \subset \mathcal{L}(\mathcal{A}_2)$? E, se $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) = \emptyset$? E ainda, se $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$? Ou ainda, se $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2) = \Sigma^\star$?

Exemplo 65 Para ilustrar o método que acabámos de expor, supomos dados os AFDs $\mathcal{A}_1$ e $\mathcal{A}_2$ seguintes.





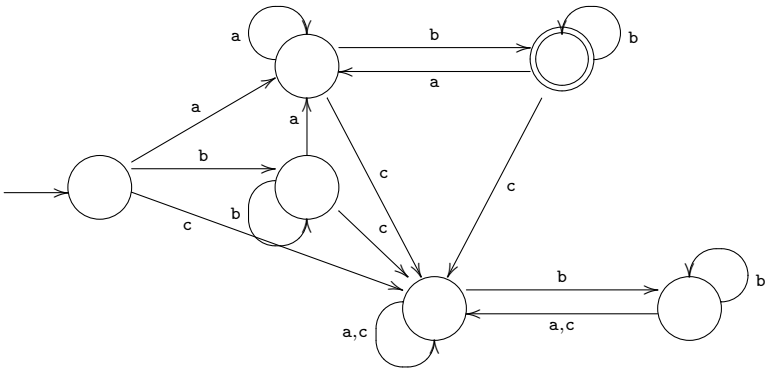
A intersecção das linguagens que estes autómatos reconhecem é o conjunto das palavras de alfabeto $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ que têm algum a, terminam em b e não têm c's. Observemos que os autómatos não têm alfabetos iguais, pelo que antes de aplicar o processo, vamos transformar o primeiro autómato, para que passe a ser um AFD com $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ também.



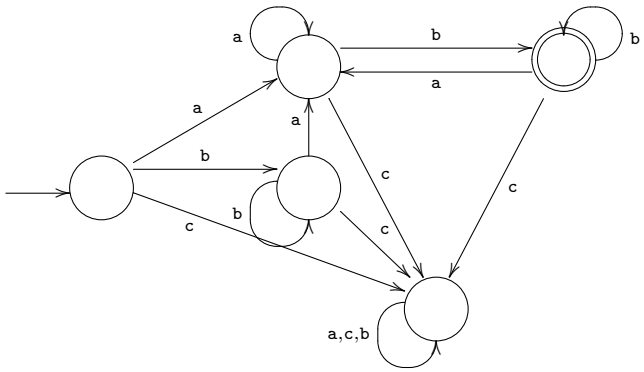
A função de transição do autómato produto é dada pela tabela seguinte e, para que reconheça $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$, o conjunto de estados finais será $\{(s_1, q_1)\}$.

	a	b	c
(s_0, q_0)	(s_1, q_0)	(s_0, q_1)	(s_2, q_0)
(s_1, q_0)	(s_1, q_0)	(s_1, q_1)	(s_2, q_0)
(s_0, q_1)	(s_1, q_0)	(s_0, q_1)	(s_2, q_0)
(s_2, q_0)	(s_2, q_0)	(s_2, q_1)	(s_2, q_0)
(s_1, q_1)	(s_1, q_0)	(s_1, q_1)	(s_2, q_0)
(s_2, q_1)	(s_2, q_0)	(s_2, q_1)	(s_2, q_0)

Tal autómato pode ser representado pelo diagrama seguinte.

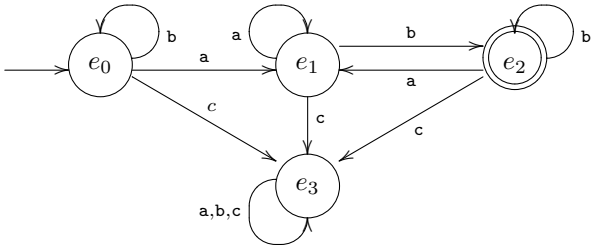


Concluimos facilmente que o AFD obtido não é mínimo. Por exemplo, o AFD seguinte é equivalente ao anterior.



Por outro lado, se notarmos que a intersecção das linguagens definidas pelos dois autómatos dados é *a linguagem constituída pelas palavras de alfabeto* $\{a, b, c\}$ *que têm algum* a*, não têm* c*'s e terminam em* b, a qual é razoavelmente

simples, não sendo díficil verificar que o AFD que tem o menor número de estados é o seguinte.



O que cada estado memoriza...

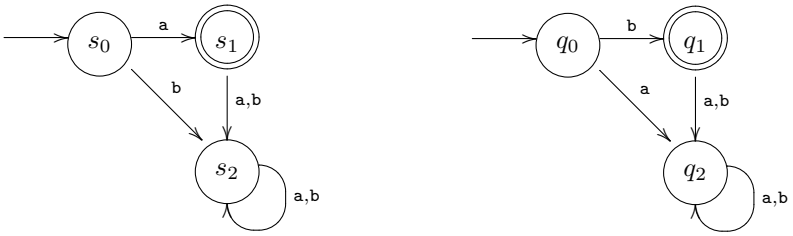
e_0: "a palavra ainda não tem a nem tem c";

e_1: "a palavra tem algum a, não tem c's e termina em a";

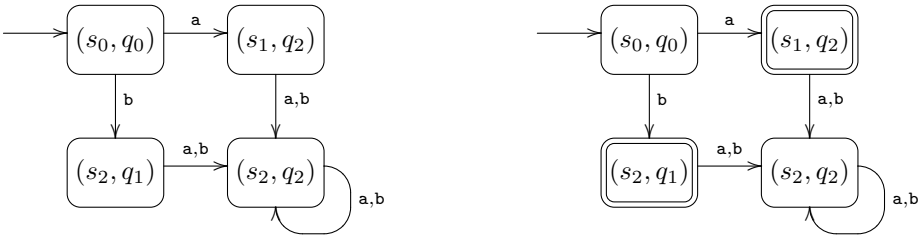
e_3: "a palavra tem algum c".

e_2: "a palavra tem algum a, termina em b e não tem c's";

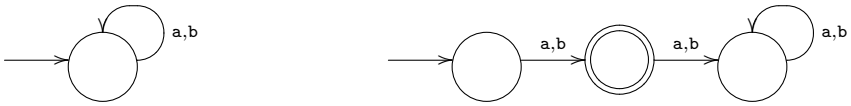
Exemplo 66 Consideremos os dois autómatos seguintes.



Com base na construção do autómato produto, obtem-se os dois autómatos seguintes, o da esquerda reconhecendo a $\{\mathtt{a}\} \cap \{\mathtt{b}\}$ (ou seja, $\{\ \}$) e o da direita $\{\mathtt{a}\} \cup \{\mathtt{b}\}$.



Claramente, este não é o melhor processo para determinar autómatos que definem $\{\ \}$ e $\{\mathtt{a}, \mathtt{b}\}$.



Contudo, como referimos, importa salientar a relevância computacional do autómato produto, pois permite dar resposta a vários problemas, sem ser necessário conhecer as linguagens que os autómatos dados aceitam.

4.3 Existência de linguagens não regulares

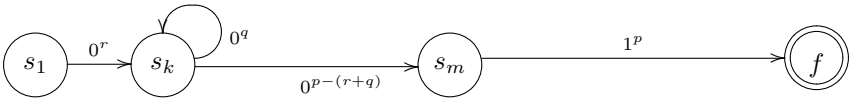
Proposição 19 *A linguagem* $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ *de alfabeto* $\Sigma = \{0, 1\}$ *não é regular.*

Prova: Por redução ao absurdo, vamos mostrar a impossibilidade de construir um autómato finito que aceite L.

Suponhamos que existia um autómato finito $\mathcal{A}$ que aceitava L. Sem perda de generalidade, supomos que $\mathcal{A}$ é um AFD. Por definição, o autómato finito $\mathcal{A}$ tem um número de estados finito. Seja p esse número e sejam $s_1, \ldots, s_p$ os estados de $\mathcal{A}$. Supomos que s_1 é o estado inicial. O autómato deveria reconhecer $0^p 1^p$. Como o autómato é determinístico, existiriam estados $s_m, f \in \{s_1, \ldots, s_p\}$ únicos, com f final, e tais que:

- s_m é o estado a que o autómato chega depois de consumir 0^p, partindo de s_1;
- f é o estado a que chega depois de consumir 1^p, partindo de s_m.

Ora, para consumir 0^p, o autómato efetua p transições nas quais, **se não fossem repetidos estados**, estariam $p + 1$ estados envolvidos. Como o autómato só tem p estados, teve que repetir algum estado para conseguir consumir 0^p. Vamos supor, por exemplo, que o estado s_k se repetiu. Então, vejamos qual pode ter sido o processamento de 0^p:



- partindo do estado inicial, o autómato consome 0^r, para um certo r tal que $0 \leq r < p$ e chega pela primeira vez a s_k (note-se que, se s_k for o estado inicial, então $r = 0$, ou seja que não teria consumido ainda nada);
- partindo de s_k consome 0^q, para algum s tal que $1 \leq q < p$, e **volta pela primeira vez** a s_k (não é importante saber se vai voltar a s_k outras vezes);
- e finalmente, ou já consumiu todos os 0's ou parte de s_k consome $0^{p-(r+q)}$ atingindo s_m.

Mas se isto acontecesse, também a palavra $0^r 0^{p-(r+q)} 1^p$, ou seja, $0^{p-q} 1^p$ seria aceite pelo autómato. O que é absurdo, pois, sendo $q \geq 1$, tem-se $0^{p-q} 1^p \notin L$. O absurdo resultou de se ter suposto que existia um autómato finito que reconhecia L. Portanto, L não é aceite por qualquer autómato finito.

Convém notar, que s_1, s_k e s_m não têm que ser distintos dois a dois, como o esquema sugere. □

A propósito... Notemos ainda que não só $0^{p-q} 1^p$ seria aceite como também $0^p 1^p$ (claro!), $0^{p+q} 1^p$, $0^{p+2q} 1^p$, $0^{p+3q} 1^p$, $0^{p+4q} 1^p$ ..., ou seja, para todo $i \in \mathbb{N}$ a palavra $0^r 0^{i \times q} 0^{p-(q+r)} 1^p$ seria aceite.

Vamos ver que também $L = \{0^p \mid p \text{ primo}\}$ não é regular. Um inteiro não negativo é **primo** sse tem dois e apenas dois divisores positivos (ele próprio e 1). Por definição, para $a, b \in \mathbb{Z}$, tem-se a **divide** b sse existir um inteiro c tal que $b = ac$. Ou seja, a divide b sse b é múltiplo de a. Para $a, b_1, b_2 \in \mathbb{Z}$, se a divide b_1 e b_2, então a divide $b_1 + b_2$ e a divide $b_1 - b_2$.

Proposição 20 *A linguagem $L = \{0^p \mid p \text{ primo}\}$ não é regular.*

Prova: Por redução ao absurdo, vamos mostrar que não existe um AFD $\mathcal{A}$ que aceite L. Vamos supor que $\mathcal{A}$ existia e ver que se chega a uma contradição. Seja $\mathcal{A}$ um AFD que reconhece L, com n estados e estado inicial s_0.

Como *o conjunto de primos é infinito* (facto que demonstraremos a seguir, c.f. Lema 3), existe um primo M tal que $M > 2n$. Pretendemos que M seja suficientemente grande quando comparado com n, bastando que seja maior do que $2n$, como ficará claro à frente. Sendo $L = \mathcal{L}(\mathcal{A})$, o autómato reconheceria a palavra 0^M. Ora, $0^M = 0^n 0^{M-n}$. Processar o prefixo 0^n envolve a passagem por $n+1$ estados e, como $\mathcal{A}$ só tem n estados, passará duas vezes por algum estado. Seja s_q, por exemplo, um estado que se repete. Como na prova anterior, supomos que

- partindo do estado inicial, o autómato consome 0^i para algum i, com $0 \leq i < n$, e chega pela primeira vez a s_q;
- a seguir, parte de s_q, consome 0^j para algum j, com $0 \leq j < n$ e $i + j \leq n$, e regressa pela primeira vez a s_q;
- finalmente, partindo do estado s_q, processa o resto da palavra, ou seja $0^{M-(i+j)}$, e chega a estado final.

Sendo $0^M = 0^i 0^j 0^{M-i-j}$ e observando que estando em s_q, o autómato processa a subpalavra 0^j e volta ao estado s_q, podemos concluir que todas as palavras

$$0^i (0^j)^k 0^{M-i-j}$$

com $k \geq 0$ seriam aceites pelo autómato. Recordemos que uma sequência de 0's pertence a L sse tem comprimento primo. Se escolhermos k convenientemente, podemos concluir que existe uma dessas palavras que é aceite pelo autómato e que não pertence à linguagem L. Como $(0^j)^k$ é a formada por repetição de 0^j exatamente k vezes, temos

$$\mid 0^i (0^j)^k 0^{M-i-j} \mid = i + kj + M - i - j = (M-j) + kj.$$

Se tomarmos $k = (M-j)$, concluimos que $(M-j) + kj = (M-j) + (M-j)j = (M-j)(j+1)$. Como inicialmente escolhemos $M > 2n$ (já a pensar nesta fatorização...), podemos garantir que $M - j \geq n + 1 \neq 1$. Por outro lado $j + 1 \geq 2$. Portanto, $(M-j)(j+1)$ não é primo, e $0^{(M-j)(j+1)}$ é aceite pelo autómato. (Absurdo!)

O absurdo resultou de se ter suposto que existia um autómato finito que aceitava L. Logo, L não é regular. □

Lema 3 *O conjunto dos inteiros não negativos que são primos é infinito.*

Prova: Suponhamos que o conjunto dos primos era finito e que tinha exatamente n elementos (sendo n um inteiro **fixo**). Suponhamos que $p_1, p_2, \ldots, p_n$ eram os primos existentes. Então, o inteiro positivo

$$1 + \prod_{i=1}^{n} p_i = 1 + p_1 p_2 \cdots p_n$$

não seria primo, pois é maior do que os primos existentes $p_1, p_2, \ldots, p_n$. Mas se $1 + p_1 p_2 \cdots p_n$ não é primo, algum dos primos o divide. Seja p_k esse primo. Como p_k divide $p_1 p_2 \cdots p_n$, se p_k dividir $1 + p_1 p_2 \cdots p_n$ então p_k divide 1.

De facto, é bem conhecido que, para $x, y, z \in \mathbb{Z}$, se x dividir y e dividir $y + z$ então também divide z. Mas, nenhum primo divide 1. Logo, o conjunto dos primos é infinito, pois o absurdo resultou de se ter suposto que o conjunto dos primos era finito. □

4.4 Lema da repetição para linguagens regulares

A técnica que usámos na demonstração das Proposição 19 e Proposição 20 vai ser fundamental na prova do Lema da Repetição para linguagens regulares, que estudaremos a seguir. Esse lema enuncia uma condição que tem de ser satisfeita por qualquer linguagem L que seja regular. Informalmente, o lema diz que:

> se L for regular e tiver palavras "suficientemente grandes", *cada uma dessas palavras tem, num certo prefixo, alguma subpalavra que pode ser repetida quantas vezes quisermos ou retirada*, sendo a palavra resultante ainda uma palavra de L. O lema define o significado de "suficientemente grande" e impõe uma condição à localização dessa subpalavra.

Proposição 21 (Lema da Repetição) *(Pumping Lemma)*

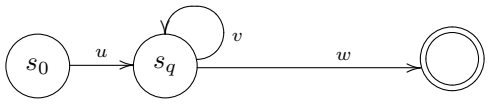
Se L é uma linguagem regular, então, existe uma constante $n \in \mathbb{N} \setminus \{0\}$, só dependente de L, tal que, se x é qualquer palavra de L com $|x| \geq n$, podemos decompor x como $x = uvw$, com $|uv| \leq n$ e $|v| \geq 1$, de modo que $uv^i w \in L$, para todo $i \geq 0$. Ou seja,

$$\exists n \in \mathbb{N} \ \ \forall x \in L \quad (|x| \geq n \Rightarrow \exists u \exists v \exists w \ \ (x = uvw \wedge |uv| \leq n \wedge v \neq \varepsilon \wedge (\forall i \in \mathbb{N} \ \ uv^i w \in L))).$$

Mais ainda, n não excede o número de estados do menor autómato finito que aceita L.

Prova: Seja L uma linguagem regular e seja $\mathcal{A}$ um AFD que reconhece L, sendo n o seu número de estados.

Consideremos uma palavra arbitrária x de L, tal que $|x| \geq n$. Um percurso no diagrama de transição de $\mathcal{A}$ que corresponda ao processamento de x envolve $|x| + 1$ estados. Como $\mathcal{A}$ tem apenas n estados, um tal percurso terá que conter algum ciclo. Seja s_q o primeiro estado que se repete e seja u o prefixo de x processado até chegar ao estado s_q. Seja v a subpalavra processada desde que sai de s_q e regressa pela primeira vez a s_q. Seja w o resto da palavra.



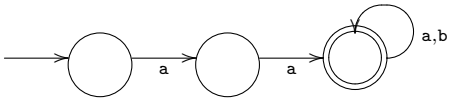
Tem-se $x = uvw$ com $|v| \geq 1$ (v corresponde a um ciclo) e $|uv| \leq n$ (caso contrário, no processamento de uv existia um outro ciclo, o que contrariava as condições sobre s_q, u, e v). Mais ainda, o autómato aceita $x = uvw$, pelo que aceita também uw, $uvvw$, $uvvvw$, ..., qualquer palavra $uv^i w$ com $i \geq 0$.

A justificação é ainda válida se o autómato com menor número de estados que aceita L for um AFND sem transições-ε. Como qualquer AFND-ε pode ser convertido num AFND com o mesmo número de estados, é verdade que n não excederia o número de estados do autómato finito com menor número de estados que aceita L. □

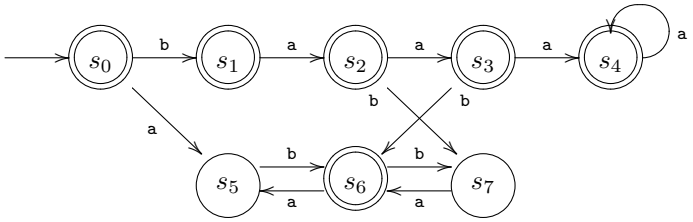
Exemplo 67 Seja L a linguagem de alfabeto $\{\mathtt{a}, \mathtt{b}\}$ que é definida pela expressão $\mathtt{aa}(\mathtt{a} + \mathtt{b})^\star$.

Vamos ver que, qualquer palavra $x \in L$ com pelo menos três símbolos, tem um prefixo uv com $|uv| \leq 3$ tal que a subpalavra v pode ser repetida quantas vezes quisermos ou retirada, originando uma palavra ainda da linguagem L.

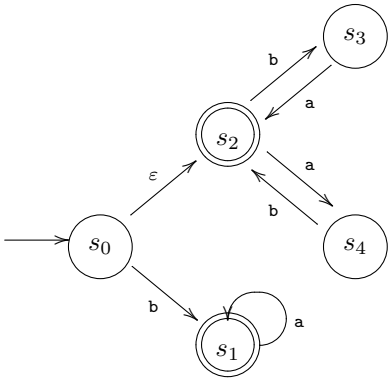
Ora, se $x \in L$ e $|x| \geq 3$ então $x = \mathtt{aaa}x'$ ou $x = \mathtt{aab}x'$, para algum $x' \in \{\mathtt{a}, \mathtt{b}\}^\star$. Se $x = \mathtt{aaa}x'$, consideramos que a subpalavra que vamos repetir ou retirar é o terceiro a. Se $x = \mathtt{aab}x'$, consideramos que a subpalavra que vamos repetir ou retirar é a formada pelo terceiro símbolo (ou seja b). O que estamos a ver é que se $x = \mathtt{aaa}x'$ e escolhermos $u = \mathtt{aa}$, $v = \mathtt{a}$ e $w = x'$, então $uv^iw \in L$, para todo $i \geq 0$. E, se for $x = \mathtt{aab}x'$ e escolhermos $u = \mathtt{aa}$, $v = \mathtt{b}$ e $w = x'$, então $uv^iw \in L$, para todo $i \geq 0$. Esta escolha satisfaz as condições $|v| \geq 1$, $|uv| \leq 3$ e $uvw = x$. Ou seja, mostrámos que $\exists n \in \mathbb{N}\ \forall x \in L\ \ (|x| \geq n \Rightarrow \exists u \exists v \exists w\ \ (x = uvw \wedge v \neq \varepsilon \wedge |uv| \leq 1 \wedge (\forall i \in \mathbb{N}\ \ uv^iw \in L)))$, tendo mostrado que podiamos tomar $n = 3$ (há de facto um autómato finito com três estados que aceita L).



Exemplo 68 A linguagem L de alfabeto $\{\mathtt{a}, \mathtt{b}\}$ definida pela expressão $\mathtt{ba}^\star + (\mathtt{ab} + \mathtt{ba})^\star$ é reconhecida pelo autómato:



Como esse autómato tem oito estados, podiamos tomar $n = 8$ para mostrar que L satisfaz a condição do Lema da Repetição. No entanto, podemos escolher $n = 5$, já que o autómato seguinte também reconhece L.



De facto, basta tomar $n = 4$, embora não exista qualquer autómato finito com menos do que cinco estados que reconheça L. Vamos ver que, para cada $x \in L$, com $|x| \geq 4$, existem u, v e w nas condições do Lema, começando por observar que:

$$\forall x \in L \ (\ |x| \geq 4 \Rightarrow \exists y \ \ (x = \texttt{baaa}y \ \vee \ x = \texttt{baba}y \ \vee \ x = \texttt{baab}y \ \vee \ x = \texttt{abab}y \ \vee \ x = \texttt{abba}y) \ )$$

Se $x = \texttt{baaa}y$, com $y \in \{\texttt{a}\}^\star$, tomamos $u = \texttt{b}$, $v = \texttt{a}$ e $w = \texttt{aa}y$, concluindo que se $x \in L$ então $\texttt{ba}^i\texttt{aa}y \in L$, para todo $i \geq 0$. Nos restantes três casos, escrevemos $x = x_1x_2x_3x_4y$, com $x_1x_2 \in \{\texttt{ab}, \texttt{ba}\}$, $x_3x_4 \in \{\texttt{ab}, \texttt{ba}\}$ e $y \in \{\texttt{ab}, \texttt{ba}\}^\star$. Vemos que, se $u = \varepsilon$, $v = x_1x_2$, e $w = x_3x_4y$ então $uv^iw = (x_1x_2)^ix_3x_4y$. Logo, se $x \in L$ então $(x_1x_2)^ix_3x_4y \in L$, para todo $i \geq 0$.

A condição do lema **nada diz sobre as palavras de L de comprimento menor que** n, que neste caso é quatro. Em particular, $\texttt{b} \in L$ não tem nenhuma subpalavra diferente de ε que possa ser repetida, pois $\texttt{b}^i \notin L$, para $i \geq 2$.

Como veremos, existem linguagens que satisfazem a condição do Lema mas não são regulares (ver Proposição 24). Mas, o Lema da Repetição indica uma condição necessária para que uma linguagem seja regular. Assim, para mostrarmos que uma dada linguagem L não é regular, podemos *tentar* provar que não satisfaz tal condição.

Para isso, temos de mostrar que **para qualquer** $n \in \mathbb{N} \setminus \{0\}$ que se tome, **existe** $x \in L$**, com** $|x| \geq n$, que não tem um prefixo uv, com $|uv| \leq n$ e $v \neq \varepsilon$, tal que a subpalavra v possa ser retirada ou repetida o número de vezes que se quiser. Em qualquer prefixo uv de x, com $|uv| \leq n$, a subpalavra v não pode ser retirada ($uv^0w \notin L$) ou existe $i \geq 2$ tal que a palavra uv^iw que resulta de repetir i vezes v em x não é de L. Como v tem de ser encontrada num prefixo de x de comprimento não superior a n, temos que arranjar uma forma de **analisar todos os prefixos de x de tamanho até** n.

Proposição 22 *A linguagem $L = \{\texttt{0}^n\texttt{1}^n \mid n \in \mathbb{N}\}$ não satisfaz a condição do Lema da Repetição e, consequentemente, não é regular.*

Prova: Seja $n \in \mathbb{N} \setminus \{0\}$ qualquer. Escolhemos $x = \texttt{0}^n\texttt{1}^n$, para que uv só tenha 0' s e $|x| \geq n$. Tem-se $x \in L$ e $|x| = 2n > n$. Quaisquer que sejam u, v e w tais que $x = uvw$ com $|v| \geq 1$ e $|uv| \leq n$, só existem 0's no prefixo uv, pois $|uv| \leq n$. Então, como $v \neq \varepsilon$, se "cortarmos" v, alteramos o número de 0's sem alterar o número de 1's. "Cortar" corresponde a tomar $i = 0$. Para $i = 0$, teriamos $uv^0w = uw$ e $uw \notin L$, já que o número de 0's em uw é $n - |v|$, o número de 1's é n, e $n - |v| \neq n$ pois $|v| \geq 1$. Logo, L não satisfaz a condição do Lema da Repetição, e por isso não é regular. □

É importante salientar que, na prova de que uma linguagem não verifica a condição do lema, a palavra x escolhida terá de depender de n. Nem o valor de n nem a decomposição uv de x podem ser concretizados. Como no exemplo

anterior, a escolha da palavra x deverá permitir reduzir o número de cenários que se terá de considerar para garantir que todas as possibilidades para uv foram analisadas.

Note-se que não recorremos explicitamente a autómatos, como fizemos nas provas das Proposições 19 e 20.

Proposição 23 *A linguagem $L = \{\mathtt{0}^p \mid p \text{ primo}\}$ não satisfaz o Lema da Repetição.*

Prova: Seja $n \in \mathbb{N} \setminus \{0\}$ (fixo mas arbitrário). Escolhemos $x = \mathtt{0}^M$ com $M > 2n$ e M primo. Tem-se $x \in L$ e $|x| = M > 2n > n$. Seja $x = uvw$ com $|v| \geq 1$ e $|uv| \leq n$. Temos que encontrar $i \geq 0$ tal que $uv^iw \notin L$, isto é, tal que $|uv^iw|$ não seja primo. Mas, $\mid uv^iw \mid = |u| + i\,|v| + |w|$ e como $|uv| \leq n$ implica $|v| \leq n$, conclui-se que $|uw| = M - |v| > n$. Então, se se escolher $i = |u| + |w|$, vem

$$\mid uv^iw \mid = |u| + i\,|v| + |w| = (|u| + |w|)(1 + |v|)$$

com $1 + |v| \geq 2$ e $|u| + |w| \geq 2$. Consequentemente, $(|u| + |w|)(1 + |v|)$ não é primo. Logo, para $i = |u| + |v|$, tem-se $uv^iw \notin L$. Portanto, L não satisfaz a condição do Lema da Repetição e por isso não é regular. □

Exercício 4.4.1 Usar o facto de a linguagem $\{\mathtt{0}^p \mid p \text{ primo}\}$ não ser regular para concluir que, qualquer que seja o alfabeto Σ, existem linguagens de alfabeto Σ não regulares.

Como referimos acima, se conseguirmos mostrar que uma linguagem L não satisfaz a condição do lema da repetição, podemos concluir que L não é regular. Mas, se a linguagem L satisfizer a condição do lema, nada se pode dizer, pois existem linguagens que satisfazem essa condição e não são regulares (a Proposição 24 apresenta uma dessas linguagens). Pelo contrário, o teorema de Myhill-Nerode, que apresentamos na secção 4.5, permite decidir se uma linguagem é ou não é regular.

Proposição 24 *A linguagem $L = \{\mathtt{0}^k \mid k \in \mathbb{N}\} \cup \{\mathtt{1}^r\mathtt{0}^{p^2} \mid r, p \in \mathbb{N} \setminus \{0\}\}$, de alfabeto $\Sigma = \{\mathtt{0}, \mathtt{1}\}$, não é regular e satisfaz a condição do lema da repetição (para lings. regulares) para $n = 1$.*

Prova: De facto, se $x \in L$ e $|x| \geq 1$ então x está num dos dois casos seguintes: $x = \mathtt{0}\mathtt{0}^k$, para algum $k \geq 0$, ou $x = \mathtt{1}\mathtt{1}^s\mathtt{0}^{q^2}$, para algum $s \geq 0$ e algum $q \geq 1$. Para $x = \mathtt{0}\mathtt{0}^k$, escolhemos $u = \varepsilon$, $v = \mathtt{0}$, e $w = \mathtt{0}^k$, e sendo $uv^iw = \mathtt{0}^{k+i}$, concluimos que $\forall i \in \mathbb{N} \;\; uv^iw \in L$. Para $x = \mathtt{1}\mathtt{1}^s\mathtt{0}^{q^2}$, escolhemos $u = \varepsilon$, $v = \mathtt{1}$, e $w = \mathtt{1}^s\mathtt{0}^{q^2}$, e vemos que, se $i \geq 1$ ou $s \geq 1$ então $uv^iw \in \{\mathtt{1}^r\mathtt{0}^{p^2} \mid r, p \in \mathbb{N} \setminus \{0\}\}$, e se $i = s = 0$ então $uv^iw \in \{\mathtt{0}\}^\star$.

Portanto, para $n = 1$ tem-se

$$\forall x \in L \;\; (|x| \geq n \Rightarrow \exists u \exists v \exists w \;\; (x = uvw \wedge v \neq \varepsilon \wedge |uv| \leq 1 \wedge (\forall i \in \mathbb{N} \;\; uv^iw \in L)))$$

Mas, L não é regular. De facto, suponhamos que L era regular e que existia um AFD que aceitava L. Como o autómato é finito, tem um conjunto de estados finito e, portanto, o conjunto de estados finais também é finito.

Assim, existem inteiros positivos p e q distintos tais que $\mathtt{1}\mathtt{0}^{p^2}$ e $\mathtt{1}\mathtt{0}^{q^2}$ levam o autómato ao mesmo estado final. Suponhamos, sem perda de generalidade, que $q > p$. Então, como o autómato é determinístico, também as palavras $\mathtt{1}\mathtt{0}^{p^2}\mathtt{0}^{(p+1)^2-p^2}$ e $\mathtt{1}\mathtt{0}^{q^2}\mathtt{0}^{(p+1)^2-p^2}$ deveriam levar o autómato ao mesmo estado. Mas, $\mathtt{1}\mathtt{0}^{p^2}\mathtt{0}^{(p+1)^2-p^2} = \mathtt{1}\mathtt{0}^{(p+1)^2}$ pertence à linguagem, e $\mathtt{1}\mathtt{0}^{q^2}\mathtt{0}^{(p+1)^2-p^2} = \mathtt{1}\mathtt{0}^{q^2+2p+1}$ não pertence à linguagem, porque, como $q > p$, tem-se

$$q^2 < q^2 + 2p + 1 < (q+1)^2 = q^2 + 2q + 1,$$

pelo que $q^2 + 2p + 1$ não é um quadrado perfeito. Conclui-se que ou $\mathtt{1}\mathtt{0}^{q^2+2p+1}$ é aceite e não é da linguagem, ou $\mathtt{1}\mathtt{0}^{(p+1)^2}$ não é aceite e é da linguagem. Tal é absurdo. Logo, o autómato que supusemos que existia não existe. □

Exercício 4.4.2 Mostrar que a linguagem das palavras de alfabeto $\{\mathtt{0}\}$ cujo comprimento é um quadrado perfeito não é regular. (Sugestão: adaptar a prova da Proposição 24).

4.5 Autómatos finitos determinísticos mínimos

Seja $\mathcal{A}$ um qualquer AFD de alfabeto Σ. Sejam x e y palavras de alfabeto Σ. Dizemos que x e y são *equivalentes do ponto de vista do autómato* $\mathcal{A}$ se e só se levam o autómato $\mathcal{A}$ ao mesmo estado. Palavras equivalentes do ponto de vista de um autómato $\mathcal{A}$ são **indistinguíveis** para $\mathcal{A}$.

Na prova da Proposição 24, para $L = \{\mathtt{0}^k \mid k \in \mathbb{N}\} \cup \{\mathtt{1}^r\mathtt{0}^{p^2} \mid r, p \in \mathbb{N} \setminus \{0\}\}$, mostrámos que, qualquer que fosse AFD que considerassemos, a palavra $\mathtt{1}\mathtt{0}^{p^2}$ não seria equivalente (do ponto de vista desse autómato) à palavra $\mathtt{1}\mathtt{0}^{q^2}$, para todo $p \geq 1$ e $q > p$. Daí concluimos que, as palavras da forma $\mathtt{1}\mathtt{0}^{p^2}$, com $p \geq 1$, eram todas não equivalentes do ponto de vista de qualquer AFD que aceitasse L. Como há um número infinito de palavras dessa forma, um tal AFD teria um número infinito de estados. Logo, por definição, não seria um autómato finito e, portanto, não existia.

Veremos que, o **AFD mínimo** que aceita uma linguagem regular L é aquele que apenas distingue palavras que teriam de levar a estados diferentes *qualquer* AFD que aceitasse L, i.e., palavras que seriam não equivalentes devido a condições intrinsecas da linguagem L (como $\mathtt{1}\mathtt{0}^{p^2}$ e $\mathtt{1}\mathtt{0}^{q^2}$, com $p \neq q$, no exemplo acima).

Vamos agora aprofundar estes conceitos de "palavras equivalentes do ponto de vista de um autómato" e de "palavras equivalentes do ponto de vista da linguagem".

Recordamos que uma relação binária $R \subseteq \Sigma^\star \times \Sigma^\star$ é de equivalência em $\Sigma^\star$ sse é reflexiva, simétrica e transitiva. Se $(x, y) \in R$, as palavras x e y dizem-se equivalentes segundo R. A classe de equivalência de x é $\{y \mid (x, y) \in R\}$. O conjunto das classes de equivalência de R designa-se por conjunto quociente e denota-se por $\Sigma^\star/R$. Uma relação de equivalência é de **índice finito** sse o seu conjunto quociente é finito.

Palavras indistinguíveis para um AFD. A noção de palavras equivalentes segundo um AFD $\mathcal{A}$ de alfabeto Σ é formalizada pela relação de equivalência $\mathcal{R}_\mathcal{A}$ em $\Sigma^\star$ definida por

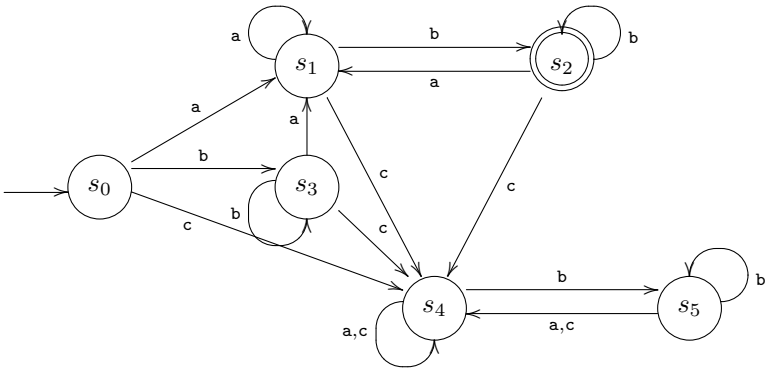
$$\mathcal{R}_\mathcal{A} = \{(x, y) \mid x, y \in \Sigma^\star \text{ e } x \text{ e } y \text{ levam o autómato } \mathcal{A} \text{ do estado inicial ao mesmo estado}\}.$$

A relação $\mathcal{R}_\mathcal{A}$ é de equivalência porque:

- $\mathcal{R}_\mathcal{A}$ é reflexiva – qualquer palavra leva $\mathcal{A}$ do estado inicial a algum estado, pelo que $\forall x \in \Sigma^\star \ \ (x, x) \in \mathcal{R}_\mathcal{A}$;
- $\mathcal{R}_\mathcal{A}$ é simétrica – se x leva $\mathcal{A}$ ao mesmo estado que y, também y leva $\mathcal{A}$ ao mesmo estado que x;
- $\mathcal{R}_\mathcal{A}$ é transitiva – se x leva $\mathcal{A}$ ao mesmo estado que y e y leva $\mathcal{A}$ ao mesmo estado que z, então x e z levam $\mathcal{A}$ ao mesmo estado, porque, sendo $\mathcal{A}$ determinístico, y não podia levar $\mathcal{A}$ a dois estados diferentes.

Denotamos por $\mathcal{C}_x$ a classe da palavra x para $\mathcal{R}_\mathcal{A}$. Claramente, *existe uma bijecção do conjunto das classes de equivalência de $\mathcal{R}_\mathcal{A}$ no conjunto dos estados de $\mathcal{A}$ que são acessíveis do estado inicial*. Cada classe de equivalência fica associada a um estado do autómato $\mathcal{A}$, sendo constituída pelo conjunto das palavras que levam o autómato do estado inicial a esse estado. Portanto, $\mathcal{R}_\mathcal{A}$ é de índice finito, pois $\mathcal{A}$ tem um número finito de estados. Os estados que não são acessíveis do estado inicial são sempre trivialmente desnecessários. No entanto, tais estados podem ocorrer em AFDs obtidos por alguns dos métodos de conversão já estudados (se não forem retirados).

Exemplo 69 Seja $\mathcal{A}$ o AFD dado pelo diagrama seguinte.



Como `aaba` e `bbaaa` levam o autómato $\mathcal{A}$ de s_0 a s_1, temos $(\texttt{aaba}, \texttt{bbaaa}) \in \mathcal{R}_\mathcal{A}$. Que outras palavras são equivalentes a estas? Analisando o diagrama, vemos que as palavras que levam $\mathcal{A}$ de s_0 a s_1 são as que terminam em `a` e não têm `c`'s. Logo, por definição de $\mathcal{R}_\mathcal{A}$, tais palavras constituem a classe de equivalência $\mathcal{C}_{\texttt{aaba}}$. Vemos também que, as palavras que levam o autómato de s_0 a s_4 são as que têm algum `c` e terminam em `a` ou `c`. Portanto, $\mathcal{C}_{\texttt{aaba}} = \mathcal{C}_{\texttt{a}} = \{x \mid x \text{ termina em } \texttt{a} \text{ e não tem } \texttt{c}\text{'s}\}$ e $\mathcal{C}_{\texttt{c}} = \{x \mid x \text{ tem algum } \texttt{c} \text{ e termina em } \texttt{a} \text{ ou } \texttt{c}\}$ são as classes de equivalência de $\mathcal{R}_\mathcal{A}$ que correspondem aos estados s_1 e s_4, e $\mathcal{R}_\mathcal{A}$ tem seis classes de equivalência.

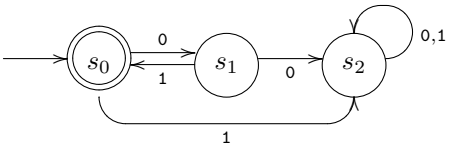
Palavras intrinsicamente indistinguíveis. "Palavras não equivalentes do ponto de vista de uma linguagem L de alfabeto Σ" são palavras que teriam de ser distinguidas por *qualquer* AFD que aceitasse L. Formalmente, a noção de palavras equivalentes do ponto de vista de L é traduzida por "equivalentes segundo $\mathcal{R}_L$", para $\mathcal{R}_L$ definida em $\Sigma^\star$ por:

$$\mathcal{R}_L = \{(x, y) \mid \forall z \in \Sigma^\star \ \ (xz \in L \Leftrightarrow yz \in L)\}.$$

Ou seja, $x, y \in \Sigma^\star$ não são equivalentes segundo $\mathcal{R}_L$ sse existir $z \in \Sigma^\star$ tal que $xz \in L \wedge yz \notin L$ ou $xz \notin L \wedge yz \in L$. Denotamos por $[x]$ a classe de equivalência da palavra x para $\mathcal{R}_L$. A relação $\mathcal{R}_L$ é de equivalência porque:

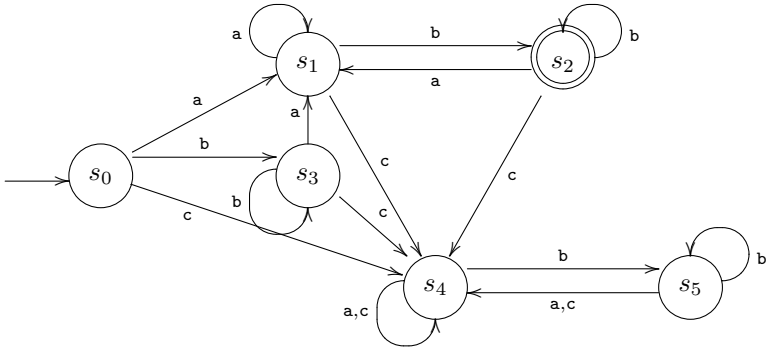
- $\mathcal{R}_L$ é reflexiva – $(x, x) \in \mathcal{R}_L$, para todo $x \in \Sigma^\star$, porque $\forall z \in \Sigma^\star (xz \in L \Leftrightarrow xz \in L)$;
- $\mathcal{R}_L$ é simétrica – se $(x, y) \in \mathcal{R}_L$ então $(y, x) \in \mathcal{R}_L$, dado que $(xz \in L \Leftrightarrow yz \in L)$ e $(yz \in L \Leftrightarrow xz \in L)$ são condições equivalentes;
- $\mathcal{R}_L$ é transitiva – se $(x, y) \in \mathcal{R}_L$ e $(y, t) \in \mathcal{R}_L$ então $(x, t) \in \mathcal{R}_L$, porque se $\forall z' \in \Sigma^\star \ (xz' \in L \Leftrightarrow yz' \in L)$ e $\forall z'' \in \Sigma^\star \ (yz'' \in L \Leftrightarrow tz'' \in L)$ então, tomando $z' = z''$, tem-se $xz \in L \Leftrightarrow tz \in L$, para todo $z \in \Sigma^\star$.

Exemplo 70 Consideremos a linguagem $L = \{\mathtt{01}\}^\star$, sobre $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. As palavras $\mathtt{0}$ e $\mathtt{00}$ não são equivalentes segundo $\mathcal{R}_L$ pois, por exemplo, para $z = \mathtt{1}$, tem-se $\mathtt{0}z = \mathtt{01} \in L$ e $\mathtt{00}z = \mathtt{001} \notin L$. Logo, $[\mathtt{00}] \neq [\mathtt{0}]$. Vemos que $\mathtt{00}z \notin L$, para todo $z \in \Sigma^\star$. Também, $\mathtt{1}z \notin L$, para todo $z \in \Sigma^\star$. Logo, $(\mathtt{00}, \mathtt{1}) \in \mathcal{R}_L$, ou seja, $[\mathtt{00}] = [\mathtt{1}]$. Podemos verificar que $\mathcal{R}_L$ tem três classes de equivalência: $[\varepsilon] = L$, a classe de $\mathtt{0}$, isto é, $[\mathtt{0}] = \{\mathtt{01}\}^\star\{\mathtt{0}\}$, e a classe de $\mathtt{00}$, que é formada pelas restantes palavras de $\Sigma^\star$, ou seja, $[\mathtt{00}] = \Sigma^\star \setminus (L \cup \{\mathtt{01}\}^\star\{\mathtt{0}\})$. Essas classes determinam os três estados do AFD mínimo que aceita L, representado abaixo, correspondendo a s_0, s_1 e s_2, respetivamente.



Se tentarmos justificar a necessidade dos estados s_0, s_1 e s_2, como fizemos nas primeiras secções, vemos que os argumentos correspondem aos que usámos para definir a equivalência (ou não equivalência) de palavras segundo $\mathcal{R}_L$.

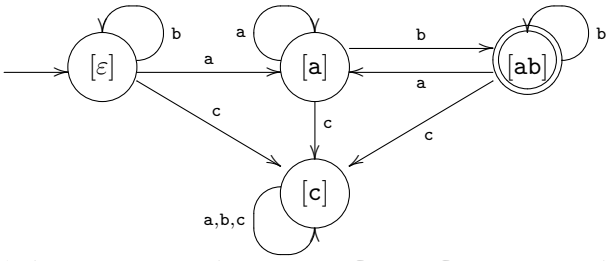
Exemplo 71 Seja L a linguagem de alfabeto $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ aceite pelo AFD $\mathcal{A}$, considerado no Exemplo 65.



Como $\mathcal{L}(\mathcal{A}) = \{x \mid x$ tem algum a, termina em b e não tem c's$\}$, as classes de equivalência de $\mathcal{R}_L$ são quatro:

- $[\mathtt{c}]$, constituída pelas palavras que têm algum c, pois se x tiver algum c, não existe $z \in \Sigma^\star$ tal que $xz \in L$;
- $[\varepsilon]$, $[\mathtt{a}]$ e $[\mathtt{ab}]$, constituídas pelas palavras que não têm c's, sendo $[\varepsilon] = \mathcal{L}(\mathtt{b}^\star)$, $[\mathtt{a}] = \mathcal{L}((\mathtt{a}+\mathtt{b})^\star\mathtt{a})$, e $[\mathtt{ab}] = \mathcal{L}((\mathtt{a}+\mathtt{b})^\star\mathtt{a}(\mathtt{a}+\mathtt{b})^\star\mathtt{b})$, pois
  - se x não tem a's, então $xz \in L$ sse z tem algum a, não tem c's e termina em b,
  - se x termina em a, então $xz \in L$ sse z termina em b e não tem c's,
  - se x termina em b e tem algum a, então $xz \in L$ sse ou $z = \varepsilon$ ou z termina em b e não tem c's,

O diagrama do AFD mínimo que aceita L é:



$[\varepsilon] = \mathcal{C}_\varepsilon \cup \mathcal{C}_\mathtt{b}$, junção de s_0 e s_3
$[\mathtt{c}] = \mathcal{C}_\mathtt{c} \cup \mathcal{C}_\mathtt{cb}$, junção de s_4 e s_5
$[\mathtt{a}] = \mathcal{C}_\mathtt{a} = [\mathtt{a}]$, mantém s_1
$[\mathtt{ab}] = \mathcal{C}_\mathtt{ab} = [\mathtt{ab}]$, mantém s_2

Se $\mathcal{A}'$ designar o AFD mínimo, tem-se $\mathcal{R}_{\mathcal{A}'} = \mathcal{R}_L$. Esta propriedade é válida para qualquer AFD e será provada mais à frente. Acima, à direita, mostramos que, na *minimização do AFD* $\mathcal{A}$, os estados determinados por palavras que não eram equivalentes segundo $\mathcal{R}_\mathcal{A}$ mas que são indistinguíveis segundo $\mathcal{R}_L$ são aglutinados num só estado de $\mathcal{A}'$.

4.5.1 Existência de um AFD mínimo para cada linguagem regular

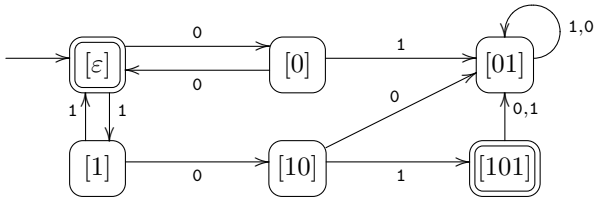
Dizemos que dois AFDs $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ e $\mathcal{A}' = (Q, \Sigma, \delta', q_0, F')$ são **isomorfos** sse forem iguais a menos dos nomes dados aos estados. Formalmente, tal significa que existe uma bijecção Ψ de S em Q, com $\delta(s,a) = s'$ sse $\delta'(\Psi(s), a) = \Psi(s')$, para todo $a \in \Sigma$ e $s, s' \in S$, e tal que $q_0 = \Psi(s_0)$ e $F' = \{\Psi(f) \mid f \in F\}$. Vamos ver que, neste sentido, o AFD mínimo para L é único.

O AFD mínimo que aceita uma dada linguagem L é caracterizado pela relação $\mathcal{R}_L$, como indica o resultado seguinte. Este resultado é um corolário imediato do Teorema de Myhill-Nerode, que provaremos na secção 4.6.

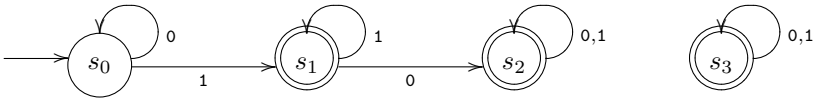
Proposição 25 *Se L é uma linguagem regular, a relação $\mathcal{R}_L$ determina o AFD mínimo que aceita L. A menos de isomorfismo, o AFD mínimo que aceita L é $\mathcal{A}_\mathtt{min} = (\Sigma^\star/\mathcal{R}_L, \Sigma, \delta, [\varepsilon], F)$, onde $\Sigma^\star/\mathcal{R}_L$ corresponde ao conjunto das classes de $\mathcal{R}_L$, sendo $F = \{[x] \mid x \in L\}$, e δ definida por $\delta([x], a) = [xa]$, para todo $x \in \Sigma^\star$ e $a \in \Sigma$.*

Exemplo 72 Para construirmos o AFD mínimo que aceita a linguagem descrita por $(\mathtt{00}+\mathtt{11})^\star(\varepsilon+\mathtt{101})$, partimos de $[\varepsilon]$ e seguimos a definição de δ para determinar os restantes estados:

$\delta([\varepsilon],\mathtt{0}) \stackrel{\text{def}}{=} [\varepsilon\mathtt{0}] = [\mathtt{0}]$	$[\varepsilon] \neq [\mathtt{0}]$ pois $\mathtt{0} \notin L$ e $\varepsilon \in L$.
$\delta([\varepsilon],\mathtt{1}) \stackrel{\text{def}}{=} [\mathtt{1}]$	$[\varepsilon] \neq [\mathtt{1}]$ porque $\mathtt{1} \notin L$ e $\varepsilon \in L$, e $[\mathtt{0}] \neq [\mathtt{1}]$ porque $\mathtt{01} \notin L$ e $\mathtt{11} \in L$.
$\delta([\mathtt{1}],\mathtt{1}) \stackrel{\text{def}}{=} [\mathtt{11}] = [\varepsilon]$	tem-se $\mathtt{1}z \in L$ sse $z \in L$. Também, $\varepsilon z \in L$ sse $z \in L$.
$\delta([\mathtt{1}],\mathtt{0}) \stackrel{\text{def}}{=} [\mathtt{10}]$	$[\mathtt{10}]$ é distinta de $[\varepsilon]$, $[\mathtt{0}]$ e $[\mathtt{1}]$ pois $(\mathtt{10},\varepsilon) \notin \mathcal{R}_L$ pois $\mathtt{10}\underline{\mathtt{1}} \in L$ e $\varepsilon\underline{\mathtt{1}} \notin L$, $(\mathtt{10},\mathtt{0}) \notin \mathcal{R}_L$ pois $\mathtt{10}\underline{\mathtt{0}} \notin L$ e $\mathtt{0}\underline{\mathtt{0}} \in L$, e $(\mathtt{10},\mathtt{1}) \notin \mathcal{R}_L$ pois $\mathtt{10}\underline{\mathtt{100}} \notin L$ e $\mathtt{1}\underline{\mathtt{100}} \notin L$.
$\delta([\mathtt{0}],\mathtt{0}) \stackrel{\text{def}}{=} [\mathtt{00}] = [\varepsilon]$	
$\delta([\mathtt{0}],\mathtt{1}) \stackrel{\text{def}}{=} [\mathtt{01}]$	$[\mathtt{01}]$ é uma nova classe, porque $\mathtt{01}z \notin L$, para todo $z \in \Sigma^\star$
$\delta([\mathtt{10}],\mathtt{0}) \stackrel{\text{def}}{=} [\mathtt{100}] = [\mathtt{01}]$	
$\delta([\mathtt{10}],\mathtt{1}) \stackrel{\text{def}}{=} [\mathtt{101}]$	$[\mathtt{101}]$ é uma nova classe, pois $\forall z \in \Sigma^\star\ (\mathtt{101}z \in L \Leftrightarrow z = \varepsilon)$
$\delta([\mathtt{101}],\mathtt{0}) \stackrel{\text{def}}{=} [\mathtt{1010}] = [\mathtt{01}]$	$\delta([\mathtt{101}],\mathtt{1}) \stackrel{\text{def}}{=} [\mathtt{1011}] = [\mathtt{01}]$



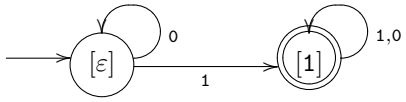
Exemplo 73 Seja $\mathcal{A}$ o AFD seguinte e $\hat{\delta}(s_0,x)$ o estado a que x leva o $\mathcal{A}$, para cada $x \in \Sigma^\star$.



A relação $\mathcal{R}_\mathcal{A}$ é traduzida por: $x\, R_\mathcal{A}\, y$ sse $\hat{\delta}(s_0,x) = \hat{\delta}(s_0,y)$. As classes de equivalência de $\mathcal{R}_\mathcal{A}$ são:

$$\begin{aligned}
\mathcal{C}_\varepsilon &= \{x \mid x \in \{\mathtt{0},\mathtt{1}\}^\star \wedge \hat{\delta}(s_0,x) = s_0\} = \{\mathtt{0}^n \mid n \geq 0\} = \mathcal{L}(\mathtt{0}^\star)\\
\mathcal{C}_\mathtt{1} &= \{x \mid x \in \{\mathtt{0},\mathtt{1}\}^\star \wedge \hat{\delta}(s_0,x) = s_1\} = \{\mathtt{0}^n\mathtt{11}^k \mid n \geq 0 \wedge k \geq 0\} = \mathcal{L}(\mathtt{0}^\star\mathtt{11}^\star)\\
\mathcal{C}_\mathtt{10} &= \{x \mid x \in \{\mathtt{0},\mathtt{1}\}^\star \wedge \hat{\delta}(s_0,x) = s_2\} = \mathcal{L}(\mathtt{0}^\star\mathtt{11}^\star\mathtt{0}(\mathtt{0}+\mathtt{1})^\star)
\end{aligned}$$

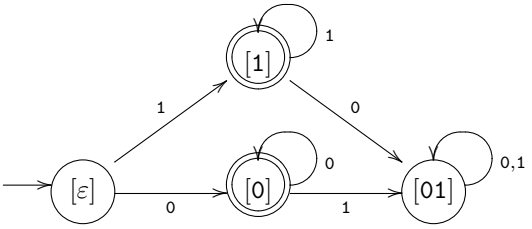
A linguagem aceite pelo autómato é dada por $\mathcal{L}(\mathcal{A}) = \mathcal{C}_\mathtt{1} \cup \mathcal{C}_\mathtt{10} = \mathcal{L}(\mathtt{0}^\star\mathtt{1}(\mathtt{0}+\mathtt{1})^\star)$. Claramente, o AFD $\mathcal{A}$ não é mínimo. Por exemplo, s_3 é dispensável, já que não é acessível de s_0. O AFD mínimo para $\mathcal{L}(\mathcal{A})$ é:



Notamos que, $[\mathtt{1}] = \mathcal{C}_\mathtt{1} \cup \mathcal{C}_\mathtt{10}$ e $[\varepsilon] = \mathcal{C}_\varepsilon$, consequência da identificação de classes de $\mathcal{R}_\mathcal{A}$ indistinguíveis segundo $\mathcal{R}_L$.

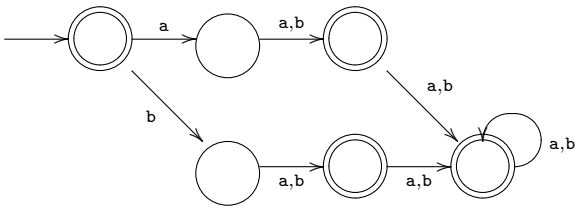
Exemplo 74 Seja $L = \mathcal{L}(0^\star + 1^\star) = \{x \mid x \in \{0,1\}^\star \text{ e } x \text{ não tem 0's ou não tem 1's}\}$. O AFD mínimo para L encontra-se à direita, sendo $\Sigma^\star/\mathcal{R}_L = \{[\varepsilon], [1], [0], [01]\}$ e $L = [0] \cup [1]$.

$$
\begin{array}{lcl}
[\varepsilon] & = & \{\varepsilon\} \\
[1] & = & \{1^k \mid k \geq 1\} \\
[0] & = & \{0^k \mid k \geq 1\} \\
[01] & = & \{x \in \{0,1\}^\star \mid x \text{ tem 0's e tem 1's}\}
\end{array}
$$

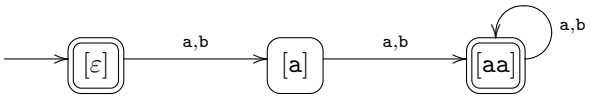


Exemplo 75 As classes de equivalência de $\mathcal{R}_\mathcal{A}$ para o AFD $\mathcal{A}$ representado, à direita são:

$$
\begin{array}{l}
\mathcal{C}_\varepsilon = \{\varepsilon\}, \quad \mathcal{C}_\mathtt{a} = \{\mathtt{a}\}, \quad \mathcal{C}_\mathtt{b} = \{\mathtt{b}\} \\
\mathcal{C}_\mathtt{aa} = \{\mathtt{aa}, \mathtt{ab}\}, \quad \mathcal{C}_\mathtt{ba} = \{\mathtt{ba}, \mathtt{bb}\} \\
\mathcal{C}_\mathtt{aaa} = \{x \mid x \in \{\mathtt{a},\mathtt{b}\}^\star, |x| \geq 3\}
\end{array}
$$



Como $\mathcal{L}(\mathcal{A}) = \{\mathtt{a},\mathtt{b}\}^\star \setminus \{\mathtt{a},\mathtt{b}\}$, o AFD mínimo para $\mathcal{L}(\mathcal{A})$ é:



Vemos que, $\mathcal{L}(\mathcal{A}) = \mathcal{C}_\varepsilon \cup \mathcal{C}_\mathtt{aa} \cup \mathcal{C}_\mathtt{ba} \cup \mathcal{C}_\mathtt{aaa} = [\varepsilon] \cup [\mathtt{aa}]$.

Temos, $[\mathtt{aa}] = \{x \mid x \in \{\mathtt{a},\mathtt{b}\}^\star, |x| \geq 2\}$, $[\varepsilon] = \{\varepsilon\}$, e $[\mathtt{a}] = \{\mathtt{a},\mathtt{b}\}$.

Notamos que, $[\mathtt{aa}] = \mathcal{C}_\mathtt{aa} \cup \mathcal{C}_\mathtt{ba} \cup \mathcal{C}_\mathtt{aaa}$, $[\mathtt{a}] = \mathcal{C}_\mathtt{a} \cup \mathcal{C}_\mathtt{b}$ e $[\varepsilon] = \mathcal{C}_\varepsilon$. Ou seja, para reconhecimento da linguagem por um AFD, basta um único estado para memorizar a informação relevante sobre as palavras de $\mathcal{C}_\mathtt{aa} \cup \mathcal{C}_\mathtt{ba} \cup \mathcal{C}_\mathtt{aaa}$. O mesmo pode ser dito sobre $\mathcal{C}_\mathtt{a} \cup \mathcal{C}_\mathtt{b}$.

Lema 4 *Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD e $L = \mathcal{L}(\mathcal{A})$. Então, para todo $x \in \Sigma^\star$, tem-se $\mathcal{C}_x \subseteq [x]$.*

Prova: Para concluir que $\mathcal{C}_x \subseteq [x]$, basta mostrar que $(x,y) \in \mathcal{R}_L$, para todo $y \in C_x$. Ora, $y \in C_x$ sse $(x,y) \in \mathcal{R}_\mathcal{A}$, pelo que se $y \in \mathcal{C}_x$ então x e y levam $\mathcal{A}$ de s_0 ao mesmo estado. Logo, para todo $z \in \Sigma^\star$, também xz e yz levam $\mathcal{A}$ a estados iguais. Portanto, $xz \in \mathcal{L}(\mathcal{A})$ se e só se $yz \in \mathcal{L}(\mathcal{A})$, para todo $z \in \Sigma^\star$. Logo, $(x,y) \in \mathcal{R}_L$. □

Exemplo 76 Do Lema 4 e da Proposição 25, concluimos que qualquer AFD que aceite a linguagem $\mathcal{L}(0^\star + 1^\star)$, analisada no Exemplo 74, tem de ter pelo menos dois estados finais e dois estados não finais. Para a linguagem $\mathcal{L}((00 + 11)^\star(\varepsilon + 101))$, considerada no Exemplo 72, concluimos que, qualquer AFD que a aceite tem pelo menos dois estados finais e quatro não finais.

4.5.2 Minimização de um AFD: Algoritmo de Moore

Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um dado AFD e L a linguagem que reconhece. Representemos por $\hat{\delta}(s, x)$ o estado em que o AFD $\mathcal{A}$ fica se consumir x partindo de s, sendo $\hat{\delta} : S \times \Sigma^\star \to S$ a função definida por $\hat{\delta}(s, \varepsilon) = s$ e $\hat{\delta}(s, xa) = \delta(\hat{\delta}(s, x), a)$, para todo $a \in \Sigma$ e $s \in S$.

Suponhamos que duas palavras x e y levam o AFD $\mathcal{A}$ do estado inicial s_0 a dois estados $s, s' \in S$, com $s \neq s'$. Tal significa que $(x, y) \notin \mathcal{R}_\mathcal{A}$. Mas, será que $(x, y) \notin R_L$? Ou seja, x e y também levariam o AFD mínimo a estados diferentes? Por análise do AFD $\mathcal{A}$, podemos decidir se x e y são equivalentes segundo R_L, com base no seguinte.

- Se $s \in F$ e $s' \notin F$, concluimos que $(x, y) \notin R_L$, pois, para $z = \varepsilon$, temos $xz \in L \wedge yz \notin L$. Analogamente, concluimos que $(x, y) \notin R_L$ se $s \notin F$ e $s' \in F$.
- Se $s, s' \in F$ ou $s, s' \notin F$, isto é, se ambos são estados finais ou ambos não são estados finais, então x e y são equivalentes segundo R_L sse $xz' \in L \Leftrightarrow yz' \in L$, para todo $z' \in \Sigma^\star \setminus \{\varepsilon\}$. Logo, $(x, y) \notin \mathcal{R}_L$ se e só se existirem $z \in \Sigma^\star$ e $a \in \Sigma$ tais que $\hat{\delta}(s, za) \in F \wedge \hat{\delta}(s', za) \notin F$ ou $\hat{\delta}(s, za) \notin F \wedge \hat{\delta}(s', za) \in F$, pois tal equivale a dizer que, com $z' = za$, apenas uma das condições $xz' \in L$ e $yz' \in L$ é satisfeita.

Definição 5 *Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD e $L = \mathcal{L}(\mathcal{A})$. Dizemos que $s, s' \in S$ são* **estados equivalentes***, e escrevemos $s \equiv s'$, se e só se ambos são estados não são acessíveis de s_0 ou existem palavras $x, y \in \Sigma^\star$ tais que $\hat{\delta}(s_0, x) = s \wedge \hat{\delta}(s_0, y) = s' \wedge (x, y) \in R_L$.*

Pelo Lema 4, a condição $\hat{\delta}(s_0, x) = s \wedge \hat{\delta}(s_0, y) = s' \wedge (x, y) \in R_L$ equivale a $\mathcal{C}_x \cup \mathcal{C}_y \subseteq [x] = [y]$. O algoritmo de Moore para minimização de AFDs, descrito abaixo, segue da Definição 5 e da observação anterior. Antes de enunciarmos a Proposição 26, que o fundamenta, introduzimos uma outra definição e dois lemas auxiliares.

Uma relação binária R em $\Sigma^\star$ é **invariante à direita para a concatenação** sse $(x, y) \in R$ implica $(xz, yz) \in R$, para todo $z \in \Sigma^\star$, com $x, y \in \Sigma^\star$ quaisquer.

Lema 5 *$\mathcal{R}_\mathcal{A}$ e $\mathcal{R}_L$ são invariantes à direita para a concatenação, para qualquer AFD $\mathcal{A}$ e linguagem $L \subseteq \Sigma^\star$.*

Prova: Considerando o modo como um AFD $\mathcal{A}$ processa palavras de $\Sigma^\star$, concluimos que se x e y levam $\mathcal{A}$ a um mesmo estado então também xz e yz levam $\mathcal{A}$ a um mesmo estado, qualquer que seja $z \in \Sigma^\star$ (este resultado é provado formalmente abaixo, c.f. Lema 6). Também, $\mathcal{R}_L$ é invariante à direita, para qualquer $L \subseteq \Sigma^\star$. Por definição de $\mathcal{R}_L$, se $(x, y) \in \mathcal{R}_L$ então $xw \in L \Leftrightarrow yw \in L$, para todo $w \in \Sigma^\star$. Tomando $w = zw'$, temos $(xz)w' \in L \Leftrightarrow (yz)w' \in L$, para todo $w' \in \Sigma^\star$. Logo, se $(x, y) \in \mathcal{R}_L$ então $(xz, yz) \in \mathcal{R}_L$, para todo $z \in \Sigma^\star$. □

Lema 6 *Quaisquer que sejam $s \in S$ e $x, z \in \Sigma^\star$, tem-se $\hat{\delta}(s, xz) = \hat{\delta}(\hat{\delta}(s, x), z)$.*

Prova: Segue da definição de δ e de $\hat{\delta}$, por indução sobre $|z|$.

(caso de base) Se $|z| = 0$ então $z = \varepsilon$ e $\hat{\delta}(s, xz) = \hat{\delta}(s, x) \stackrel{\text{def}}{=} \hat{\delta}(\hat{\delta}(s, x), \varepsilon)$, para todo $x \in \Sigma^\star$ e todo $s \in S$.

(hereditariedade) Para n fixo, suponhamos que, para qualquer $z \in \Sigma^\star$ com $|z| = n$, se tem $\hat{\delta}(s, xz) = \hat{\delta}(\hat{\delta}(s, x), z)$, para todo $x \in \Sigma^\star$ e $s \in S$. E, vamos ver que então podemos concluir que para $z' \in \Sigma^\star$ com $|z'| = n + 1$ se tem uma condição análoga. Ora, se $|z'| = n + 1$ então $z' = za$, para algum z, com $|z| = n$, e algum $a \in \Sigma$. Por definição,

$$\hat{\delta}(s, xz') = \hat{\delta}(s, xza) = \hat{\delta}(s, (xz)a) \stackrel{\text{def}}{=} \delta(\hat{\delta}(s, xz), a).$$

Mas, por hipótese de indução, $\hat{\delta}(s, xz) = \hat{\delta}(\hat{\delta}(s, x), z)$. Logo, $\delta(\hat{\delta}(s, xz), a) = \delta(\hat{\delta}(\hat{\delta}(s, x), z), a)$. Mas, por definição de $\hat{\delta}$, e atendendo a que podemos representar $\hat{\delta}(s, x)$ por algum s', $\delta(\hat{\delta}(\hat{\delta}(s, x), z), a) \stackrel{\text{def}}{=} \hat{\delta}(\hat{\delta}(s, x), za) = \hat{\delta}(\hat{\delta}(s, x), z')$. Provámos assim que, se $\forall x \in \Sigma^\star \ \forall s \in S \quad \hat{\delta}(s, xz) = \hat{\delta}(\hat{\delta}(s, x), z)$, para todo z com $|z| = n$, então, para todo z' com $|z'| = n + 1$, tem-se $\forall x \in \Sigma^\star \ \forall s \in S \quad \hat{\delta}(s, xz') = \hat{\delta}(\hat{\delta}(s, x), z')$.

Portanto, por indução matemática podemos concluir que a propriedade se verifica para todo $n \in \mathbb{N}$, isto é, podemos concluir que $\forall z \in \Sigma^\star \ \forall x \in \Sigma^\star \ \forall s \in S \quad \hat{\delta}(s, xz) = \hat{\delta}(\hat{\delta}(s, x), z)$. □

Proposição 26 *Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD. Então, quaisquer que sejam os estados s e s' acessíveis de s_0, tem-se: (i) se $s \in F$ e $s' \notin F$, então $s \not\equiv s'$ e, analogamente, se $s \notin F$ e $s' \in F$, então $s \not\equiv s'$; e (ii) se ambos são finais ou ambos não finais, isto é, se $s, s' \in F$ ou $s, s' \notin F$, então $s \not\equiv s'$ sse $\delta(s, a) \not\equiv \delta(s', a)$, para algum $a \in \Sigma$.*

Prova: Começamos por notar que, se $s = s'$ então $s \equiv s'$. Assim, assumimos que $s \neq s'$ e supomos que x e y são tais que $\hat{\delta}(s_0, x) = s$ e $\hat{\delta}(s_0, y) = s'$. Como *(i)* é trivial, vamos provar *(ii)*.

Se $\delta(s, a) \not\equiv \delta(s', a)$, para algum $a \in \Sigma$, então $(xa, ya) \notin R_L$, pelo Lema 4 e definição de $\equiv$. Logo, $s \not\equiv s'$, pois, se $s \equiv s'$ então $(x, y) \in R_L$ e, por $\mathcal{R}_L$ ser invariante à direita para a concatenação, teriamos $(xa, ya) \in \mathcal{R}_L$.

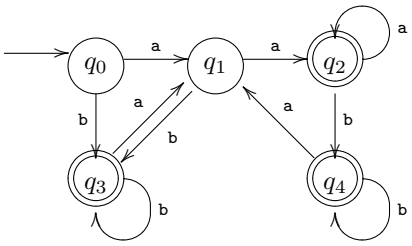
Reciprocamente, se $s \not\equiv s'$ então $(x, y) \notin \mathcal{R}_L$ e, consequentemente, existe $z \in \Sigma^\star$ tal que $xz \in L \wedge yz \notin L$ ou $xz \notin L \wedge yz \in L$. Como s e s' são ambos finais ou ambos não finais, $|z| \geq 1$. Então $z = az'$, para algum $a \in \Sigma$, e $\delta(s, a) \not\equiv \delta(s', a)$. Caso contrário, $(xa, ya) \in R_L$ e, portanto, para z' teriamos $(xa)z' \in L \Leftrightarrow (ya)z' \in L$. □

Algoritmo de Moore para minimizar um AFD

Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD. Começamos por retirar todos os estados não acessíveis de s_0. Para facilitar a descrição, supomos que $S' = \{s_0, s_1, \ldots, s_m\}$ é o conjunto dos estados de $\mathcal{A}$ acessíveis de s_0. Como a relação $\equiv$ é simétrica, basta considerar (s_i, s_j), para $0 \leq i \leq j \leq m$. Para visualização, construimos uma tabela, onde os símbolos =, `X` e `?` denotam $\equiv$, $\not\equiv$ e decisão pendente. Na aplicação do algoritmo, cada par pode ter uma lista de pares pendentes associada. Inicialmente, essas listas são vazias. Assinalamos com = todas as entradas (s_i, s_i), para todo i, e preenchemos as restantes entradas assim.

- Para todo (s_i, s_j), com $s_i \in F \wedge s_j \notin F$ ou $s_i \in F \wedge s_j \notin F$, assinalar $s_i \not\equiv s_j$, colocando X em (s_i, s_j).
- Para $1 \leq j \leq m$ e $0 \leq i < j$, se (s_i, s_j) não contém X, averiguar se já é conhecido que $\delta(s_i, a) \not\equiv \delta(s_j, a)$, para algum $a \in \Sigma$ (por verificação da existência de X na entrada correspondente).
  - Se for, registar $s_i \not\equiv s_j$, assinalando (s_i, s_j) com X, e propagar a informação a todos os pares que estiverem na lista de pendentes de (s_i, s_j). *Propagar* significa assinalar com X cada um dos pares nessa lista e, recursivamente, propagar aos pares que tiverem nas listas de pendentes desses.
  - Se não for e existir $a \in \Sigma$, tal que $(\delta(s_i, a), \delta(s_j, a))$ ainda não está decidido (obviamente, sem incluir o próprio (s_i, s_j)), então, para cada $(\delta(s_i, a), \delta(s_j, a))$ sem marcação =, acrescentar (s_i, s_j) à lista de pendentes de $(\delta(s_i, a), \delta(s_j, a))$ e assinalar a entrada (s_i, s_j) com ?.
  - Caso contrário, registar $s_i \equiv s_j$, assinalando (s_i, s_j) com =.
- No final, substituir ? por = nas entradas que se mantiverem pendentes (cada entrada que não tem X, corresponde a um par de estados equivalentes).
- O conjunto de estados do AFD mínimo $\mathcal{A}'$ equivalente ao AFD $\mathcal{A}$ corresponde ao conjunto de classes de equivalência de $\equiv$ (restrita a $S' = \{s_0, s_1 \ldots, s_m\} \subseteq S$). Se $[s]$ denotar a classe do estado s, então a função de transição δ' é dada por $\delta'([s], a) = \delta(s, a)$, para todo $a \in \Sigma$. O estado inicial de $\mathcal{A}'$ é $[s_0]$ e o conjunto de estados finais é $F' = \{[s] \mid s \in F \cap S'\}$.

Exemplo 77 Vamos aplicar o algoritmo de Moore para verificar se o AFD representado é mínimo.



q_0	=				
q_1		=			
q_2	X	X	=		
q_3	X	X		=	
q_4	X	X			=
	q_0	q_1	q_2	q_3	q_4

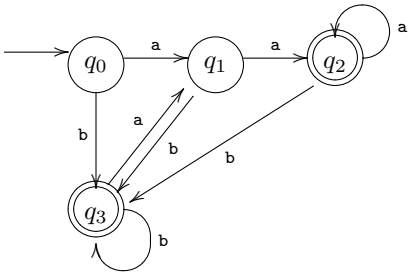
Os estados finais não são equivalentes aos estados não finais e, por isso, começamos por assinalar todos esses pares (final/não final) como distintos. Para os pares que sobram, temos:

- $\delta'(q_0, \mathtt{a}) = q_1$ e $\delta'(q_1, \mathtt{a}) = q_2$ e já sabemos que q_1 e q_2 não são equivalentes. Logo, q_0 não é equivalente a q_1.
- $\delta'(q_2, \mathtt{a}) = q_2$ e $\delta'(q_3, \mathtt{a}) = q_1$ e sabemos que q_2 e q_1 não são equivalentes. Logo, q_2 e q_3 não são equivalentes.
- Por razão análoga, q_2 e q_4 não são equivalentes, pois $\delta'(q_2, \mathtt{a}) = q_2$ e $\delta'(q_4, \mathtt{a}) = q_1$.

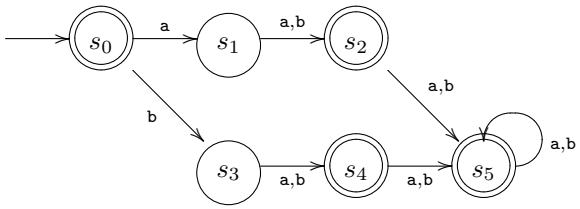
- Finalmente, para (q_3, q_4) tem-se $\delta'(q_3, \mathtt{a}) = q_1$ e $\delta'(q_4, \mathtt{a}) = q_1$ e $\delta'(q_3, \mathtt{b}) = q_3$ e $\delta'(q_4, \mathtt{b}) = q_4$, pelo que, não sendo possível distinguir q_3 de q_4, concluimos que são equivalentes.

A tabela final e o AFD mínimo estão abaixo. No AFD mínimo, usámos q_3 como representante da classe de q_3 e q_4.

q_0	$=$				
q_1	X	$=$			
q_2	X	X	$=$		
q_3	X	X	X	$=$	
q_4	X	X	X	$=$	$=$
	q_0	q_1	q_2	q_3	q_4



Exemplo 78 Por aplicação do algoritmo de Moore, vamos averiguar se o AFD representado é mínimo.



s_0	$=$					
s_1	X	$=$				
s_2		X	$=$			
s_3	X		X	$=$		
s_4		X		X	$=$	
s_5		X		X		$=$
	s_0	s_1	s_2	s_3	s_4	s_5

A tabela inicial encontra-se acima. Para os restantes pares, tem-se:

(s_0, s_2) : $s_0 \not\equiv s_2$ porque $\delta(s_0, \mathtt{a}) = s_1 \not\equiv s_5 = \delta(s_2, \mathtt{a})$.

(s_0, s_4) : $s_0 \not\equiv s_4$ porque $\delta(s_0, \mathtt{a}) = s_1 \not\equiv s_5 = \delta(s_4, \mathtt{a})$.

(s_0, s_5) : $s_0 \not\equiv s_5$ porque $\delta(s_0, \mathtt{a}) = s_1 \not\equiv s_5 = \delta(s_5, \mathtt{a})$.

(s_1, s_3) : $\delta(s_1, \mathtt{a}) = s_2 = \delta(s_1, \mathtt{b})$ e $\delta(s_3, \mathtt{a}) = s_4 = \delta(s_3, \mathtt{b})$.
Fica pendente. Assinalar dependência em (s_2, s_4).

(s_2, s_4) : $s_2 \equiv s_4$ pois $\delta(s_2, \mathtt{a}) = \delta(s_4, \mathtt{a})$ e $\delta(s_2, \mathtt{b}) = \delta(s_4, \mathtt{b})$.

(s_2, s_5) : $s_2 \equiv s_5$ pois $\delta(s_2, \mathtt{a}) = \delta(s_5, \mathtt{a})$ e $\delta(s_2, \mathtt{b}) = \delta(s_5, \mathtt{b})$.

(s_4, s_5) : $s_4 \equiv s_5$ pois $\delta(s_4, \mathtt{a}) = \delta(s_5, \mathtt{a})$ e $\delta(s_4, \mathtt{b}) = \delta(s_5, \mathtt{b})$.

s_0	$=$					
s_1	X	$=$				
s_2	X	X	$=$			
s_3	X	$?$	X	$=$		
s_4	X	X	$\overset{(s_1,\,s_3)}{=}$	X	$=$	
s_5	X	X	$=$	X	$=$	$=$
	s_0	s_1	s_2	s_3	s_4	s_5

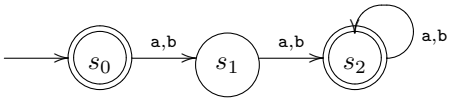
Se seguirmos o algoritmo passo a passo, a decisão (s_1, s_3) fica pendente até ao fim, pois não há informação global sobre o número de pares que podem decidir a não equivalência de (s_1, s_3). Apenas no passo final se troca $?$ por $=$.

s_0	$=$					
s_1	X	$=$				
s_2	X	X	$=$			
s_3	X	$=$	X	$=$		
s_4	X	X	$=$	X	$=$	
s_5	X	X	$=$	X	$=$	$=$
	s_0	s_1	s_2	s_3	s_4	s_5

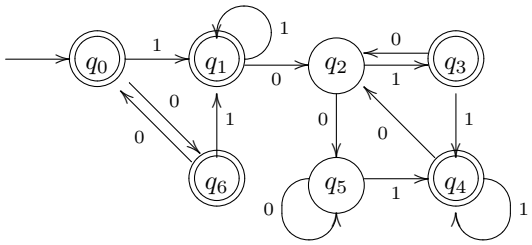
As classes de eqivalência de $\equiv$ são:

$$\{s_0\}, \{s_1, s_3\}, \{s_2, s_4, s_5\}.$$

O AFD mínimo equivalente ao AFD dado é:



Exemplo 79 Por aplicação do algoritmo de Moore, vamos minimizar o AFD representado abaixo.



A tabela inicial encontra-se à esquerda e a final à direita. Apresentamos abaixo os passos intermédios.

q_0	$=$					
q_1		$=$				
q_2	**X**	**X**	$=$			
q_3			**X**	$=$		
q_4			**X**		$=$	
q_5	**X**	**X**		**X**	**X**	$=$
q_6			**X**			**X**	$=$
	q_0	q_1	q_2	q_3	q_4	q_5	q_6

q_0	$=$					
q_1	X	$=$				
q_2	**X**	**X**	$=$			
q_3	X	?$=$	**X**	$=$		
q_4	X	(q_1, q_3) $=$	**X**	(q_2, q_5) $=$	$=$	
q_5	**X**	**X**	?$=$	**X**	**X**	$=$	
q_6	$=$	X	**X**	X	X	**X**	$=$
	q_0	q_1	q_2	q_3	q_4	q_5	q_6

(q_0, q_1) : $q_0 \not\equiv q_1$ porque $\delta(q_0, 0) = q_6 \not\equiv q_2 = \delta(q_1, 0)$.

(q_0, q_3) : $q_0 \not\equiv q_3$ porque $\delta(q_0, 0) = q_6 \not\equiv q_2 = \delta(q_3, 0)$.

(q_0, q_4) : $q_0 \not\equiv q_4$ porque $\delta(q_0, 0) = q_6 \not\equiv q_2 = \delta(q_4, 0)$.

(q_0, q_6) : $q_0 \equiv q_6$ pois são indistinguíveis já que $\delta(q_0, 1) = \delta(q_6, 1)$ e $\delta(q_0, 0) = q_6$ e $\delta(q_6, 0) = q_0$.

(q_1, q_3) : $q_1 \equiv q_3$ se e só se $q_1 \equiv q_4$ pois $\delta(q_1, 0) = \delta(q_3, 0)$ e $\delta(q_1, 1) = q_1$ e $\delta(q_3, 1) = q_4$.

Fica pendente (?) e registamos (q_1, q_3) na entrada de (q_1, q_4), para recordar a dependência.

(q_1, q_4) : $q_1 \equiv q_4$ pois são indistinguíveis já que $\delta(q_1, 0) = \delta(q_4, 0)$ e $\delta(q_4, 1) = q_4$ e $\delta(q_1, 1) = q_1$.
Podemos concluir que $q_1 \equiv q_3$, pois tal decisão só dependia de (q_1, q_4).

(q_1, q_6) : $q_1 \not\equiv q_6$ pois $\delta(q_1, 0) = q_2 \not\equiv q_0 = \delta(q_6, 0)$,

(q_2, q_5) : $q_2 \equiv q_5$ se e só se $q_3 \equiv q_4$ pois $\delta(q_2, 0) = \delta(q_5, 0)$ e $\delta(q_2, 1) = q_3$ e $\delta(q_5, 1) = q_4$.
Fica pendente (?) e registamos (q_2, q_5) na entrada de (q_3, q_4), para recordar a dependência.

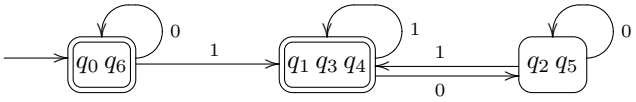
(q_3, q_4) : $q_3 \equiv q_4$ pois são indistinguíveis já que $\delta(q_3, 0) = \delta(q_4, 0)$ e $\delta(q_3, 1) = \delta(q_4, 1)$.
Podemos concluimos que $q_2 \equiv q_5$, pois tal decisão só dependia de (q_3, q_4).

(q_3, q_6) : $q_3 \not\equiv q_6$ pois já que $\delta(q_3, 0) = q_2 \not\equiv q_0 = \delta(q_6, 0)$.

(q_4, q_6) : $q_4 \not\equiv q_6$ porque $\delta(q_4, 0) = q_2 \not\equiv q_0 = \delta(q_6, 0)$.

Se seguirmos o algoritmo passo a passo, as decisões (q_1, q_3) e (q_2, q_5) ficariam pendentes até ao fim, pois não há informação global sobre o número de pares que podem decidir a não equivalência de um par. De acordo com a descrição do algoritmo, quando, por exemplo, se descobre que $q_1 \equiv q_4$, essa informação não é propagada a (q_1, q_3). Apenas no passo final se trocaria ? por $=$.

Concluimos que $q_0 \equiv q_6$, $q_1 \equiv q_3 \equiv q_4$ e $q_2 \equiv q_5$. Portanto, o AFD mínimo equivalente ao AFD dado é:



4.6 Caraterização das linguagens regulares: Teorema de Myhill-Nerode

Lema 7 *Se L é uma linguagem regular, então L é união de classes de equivalência de alguma relação de equivalência invariante à direita e de índice finito.*

Prova: Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD tal que $L = \mathcal{L}(\mathcal{A})$. Sem perda de generalidade, assumimos que todos os estados de $\mathcal{A}$ são acessíveis de s_0 (pois, teriamos um AFD equivalente se retirassemos os estados não acessíveis). A relação $\mathcal{R}_\mathcal{A}$ é de índice finito porque tem exatamente $|S|$ classes e, pelo Lema 5, é invariante à direita. A linguagem L é união das classes de equivalência de $\mathcal{R}_\mathcal{A}$ determinadas pelos estados finais de $\mathcal{A}$. □

Lema 8 *Seja L uma linguagem regular sobre Σ e seja $\mathcal{A}$ um AFD que aceita L. Então, $|\Sigma^\star/\mathcal{R}_L| \leq |\Sigma^\star/\mathcal{R}_\mathcal{A}|$. Logo, se L regular então $\mathcal{R}_L$ é de índice finito.*

Prova: Pelo Lema 4, para qualquer AFD $\mathcal{A}$, temos $\mathcal{C}_x \subseteq [x]$, para todo $x \in \Sigma^\star$, onde $\mathcal{C}_x$ designa a classe de x para $\mathcal{R}_\mathcal{A}$ e $[x]$ a sua classe para $\mathcal{R}_L$, sendo $L = \mathcal{L}(\mathcal{A})$. Como cada classe de $\mathcal{R}_\mathcal{A}$ está contida em alguma classe

de $\mathcal{R}_L$, concluimos que a relação $\mathcal{R}_L$ parte $\Sigma^\star$ em ou menos ou exatamente nas mesmas classes que $\mathcal{R}_\mathcal{A}$. Ou seja, $|\Sigma^\star/\mathcal{R}_L| \leq |\Sigma^\star/\mathcal{R}_\mathcal{A}|$. Portanto, como $\Sigma^\star/\mathcal{R}_\mathcal{A}$ é finito, $\Sigma^\star/\mathcal{R}_L$ é finito. Sabemos que se uma linguagem é regular sse existe algum AFD que a aceita. Portanto, se L é regular, $\mathcal{R}_L$ é de índice finito. □

Pelo Lema 5, sabemos que $\mathcal{R}_L$ é invariante à direita, qualquer que seja a linguagem L de alfabeto Σ. Para L regular, esta propriedade é crucial para a caraterização do AFD mínimo que aceita L, dada na Proposição 25. De facto, permite **garantir que δ fica bem definida**, quando se escreve $\delta([x], a) = [xa]$, para todo $x \in \Sigma^\star$ e $a \in \Sigma$.

Como $\mathcal{R}_L$ é invariante à direita, podemos afirmar que, quaisquer que sejam $x, y \in \Sigma^\star$ e $a \in \Sigma$, se e $[x] = [y]$ então $[xa] = [ya]$. Assim, ao dizermos que $\delta([x], a)$ é $[xa]$, definimos uma função de $\Sigma^\star/\mathcal{R}_L \times \Sigma$ em $\Sigma^\star/\mathcal{R}_L$.

Recordamos que se f é uma função de um conjunto A num conjunto B, então todo $a \in A$ tem uma e uma só imagem em B. Para se ter uma função de $\Sigma^\star/\mathcal{R}_L \times \Sigma$ em $\Sigma^\star/\mathcal{R}_L$ é necessário garantir que, quaisquer que sejam $a \in \Sigma$ e a classe X, a imagem de (X, a) seja única. Mas, X aparece representada por um elemento $x \in X$, pelo que é necessário garantir que a imagem não depende do representante escolhido. Por outras palavras, se x e y estão ambos em X, então $X = [x] = [y]$ e $[ya]$ não pode ser diferente de $[xa]$, senão a imagem de (X, a) dependia do elemento que se estava a usar para representar a classe. A invariância à direita de $\mathcal{R}_L$ garante que se $[x] = [y]$ então $[xa] = [ya]$.

O Teorema de Myhill-Nerode apresenta uma condição necessária e suficiente para que uma linguagem L de alfabeto Σ seja regular. Simultaneamente, permite provar a Proposição 25, enunciada anteriormente, que carateriza o AFD mínimo que reconhece uma linguagem regular L dada.

Proposição 27 *(***Teorema de Myhill-Nerode)**

As três afirmações seguintes, sobre uma linguagem L de $\Sigma^\star$, são equivalentes:

(i) L é aceite por um autómato finito.

(ii) L é união de classes de equivalência de alguma relação de equivalência invariante à direita e de índice finito.

(iii) A relação de equivalência $\mathcal{R}_L$ é de índice finito.

Prova: Vamos mostrar que (i) ⇒ (ii) ⇒ (iii) ⇒ (i) para concluir a equivalência das três afirmações.

- (i) ⇒ (ii). Esta implicação já foi provada (c.f., Lema 7).
- (ii) ⇒ (iii). Vamos mostrar que se L é união de classes de equivalência de alguma relação de equivalência R, invariante à direita e de índice finito, então cada classe de equivalência de R está contida em alguma classe de equivalência de $\mathcal{R}_L$. Sejam $x, y \in \Sigma^\star$ quaisquer, e tais que $(x, y) \in R$. Então, como R é invariante à direita, $(xz, yz) \in R$, para todo $z \in \Sigma^\star$. Logo, para cada z, as palavras xz e yz pertencem à mesma classe

de equivalência de R. Por outro lado, L é união de classes de equivalência de R. Assim, xz pertence a uma classe de R que está contida em L se e só se yz pertence a uma classe de R que está contida em L. Ou seja, $xz \in L \Leftrightarrow yz \in L$. Como z é qualquer, então $(x, y) \in \mathcal{R}_L$. Mostrámos que $(x, y) \in R \Rightarrow (x, y) \in \mathcal{R}_L$, para $x, y \in \Sigma^\star$ quaisquer. Logo, cada classe de R está contida nalguma classe de $\mathcal{R}_L$ e, como R é de índice finito, também $\mathcal{R}_L$ é de índice finito.

- (iii) $\Rightarrow$ (i). Sendo $\Sigma^\star/\mathcal{R}_L$ finito, podemos definir um AFD $\mathcal{A} = (\Sigma^\star/\mathcal{R}_L, \Sigma, \delta, [\varepsilon], F)$, em que o conjunto de estados é $\Sigma^\star/\mathcal{R}_L$, o estado inicial $[\varepsilon]$ é a classe da palavra vazia, o conjunto de estados finais F é o conjunto das classes de $\mathcal{R}_L$ cuja união é a linguagem L, e a função de transição é dada por $\delta([x], a) = [xa]$, quaisquer que sejam $x \in \Sigma^\star$ e $a \in \Sigma$, onde $[x]$ denota a classe de equivalência de x.

  Já vimos que δ é uma função. Por indução sobre $|y|$, não é difícil mostrar que $\hat{\delta}([x], y) = [xy]$, para todo $y \in \Sigma^\star$ e todo $x \in \Sigma^\star$. Desta igualdade obtém-se $\hat{\delta}([\varepsilon], y) = [y]$, e consequentemente y é aceite pelo autómato se e só se $y \in L$ (atendendo à definição de F). Portanto, $\mathcal{L}(\mathcal{A}) = L$. □

Como referimos, a Proposição 25 é um corolário do Teorema de Myhill-Nerode. Da prova de *(iii) $\Rightarrow$ (i)*, segue $L = \mathcal{L}(\mathcal{A}_{\texttt{min}})$. De *(ii) $\Rightarrow$ (iii)*, ou do Lema 8, concluimos que o número de estados de $\mathcal{A}_{\texttt{min}}$ não excede o número de estados de qualquer AFD que aceite L, o que justifica a afirmação de "ser o mínimo".

O teorema de Myhill-Nerode diz que uma linguagem L de alfabeto Σ é regular se e só se $\mathcal{R}_L$ é de índice finito, ou seja, $\Sigma^\star/\mathcal{R}_L$ é finito. Assim, dada uma linguagem L, se mostrarmos que o conjunto das classes de equivalência de $\mathcal{R}_L$ é infinito, concluimos que L não é regular.

Exemplo 80 A linguagem $L = \{\texttt{0}^n\texttt{1}^n \mid n \geq 0\}$ de alfabeto $\{\texttt{0}, \texttt{1}\}$ não é regular.

Vamos provar que o conjunto das classes de equivalência de $\mathcal{R}_L$ é infinito. Para isso, vamos justificar que $(\texttt{0}^p, \texttt{0}^q) \notin \mathcal{R}_L$, para $p, q \in \mathbb{N}$, com $p \neq q$. Tal permite deduzir que $[\varepsilon]$, $[\texttt{0}]$, $[\texttt{00}]$, $[\texttt{000}]$, $[\texttt{0000}]$, ... são distintas duas a duas. A classe da palavra $\texttt{0}^k$ é constituída apenas pela palavra $\texttt{0}^k$, para $k \geq 0$. Logo, $\mathcal{R}_L$ não é de índice infinito. Se $p \neq q$ então $(\texttt{0}^p, \texttt{0}^q) \notin \mathcal{R}_L$, porque existe $z \in \{\texttt{0}, \texttt{1}\}^\star$ tal que $\texttt{0}^p z \notin L$ e $\texttt{0}^q z \in L$. Basta tomar $z = \texttt{1}^q$. □

Exemplo 81 A linguagem $L = \{\texttt{0}^{n^2} \mid n \geq 0\}$ de alfabeto $\{\texttt{0}\}$ não é regular.

Vamos provar que $\Sigma^\star/R_L$ é infinito. Para isso, vamos justificar que $(\texttt{0}^{p^2}, \texttt{0}^{q^2}) \notin \mathcal{R}_L$, para $p, q \in \mathbb{N}$, com $p < q$. Tal permite permite deduzir que $[\texttt{0}^{k^2}] = \{\texttt{0}^{k^2}\}$, para $k \geq 0$, e, portanto, o conjunto das classes de $\mathcal{R}_L$ não é finito. Se $p < q$ então $(\texttt{0}^{p^2}, \texttt{0}^{q^2}) \notin \mathcal{R}_L$, porque existe $z \in \{\texttt{0}\}^\star$ tal que $\texttt{0}^{p^2} z \in L$ e $\texttt{0}^{q^2} z \notin L$. Basta tomar $z = \texttt{0}^{2p+1}$. Tem-se $\texttt{0}^{p^2} z = \texttt{0}^{(p+1)^2} \in L$, mas $\texttt{0}^{q^2} z = \texttt{0}^{q^2+2p+1} \notin L$, pois $q^2 < q^2 + 2p + 1 < (q+1)^2$ □

Exemplo 82 A linguagem das expressões regulares sobre $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ não é regular.

Para não confundir a palavra vazia com expressão ε, denotaremos o símbolo ε do alfabeto por $\boxed{\varepsilon}$. Tal linguagem pode ser definida indutivamente como uma linguagem L de alfabeto $\{\mathtt{)}, \mathtt{(}, +, ^\star, \boxed{\varepsilon}, \emptyset\} \cup \Sigma$, por: *(i)* $\mathtt{a} \in L$, $\mathtt{b} \in L$, $\boxed{\varepsilon} \in L$, $\emptyset \in L$, e *(ii)* $\mathtt{(}r^\star\mathtt{)} \in L$, $\mathtt{(}rs\mathtt{)} \in L$, e $\mathtt{(}r{+}s\mathtt{)} \in L$, quaisquer que sejam $r, s \in L$.

Tomemos expressões regulares r_k da forma $\mathtt{(((((}\cdots\mathtt{(}(\mathtt{ab})\,\mathtt{a})\,\mathtt{a})\,\mathtt{a})\cdots\mathtt{a})\,\mathtt{a})\,\mathtt{a})$ ou, mais formalmente, $r_k = \mathtt{(}^{k+1}\mathtt{ab})\,s^k$, onde $s = \mathtt{a})$, sendo $\mathcal{L}(r_k) = \{\mathtt{aba}^k\}$. Como no Exemplo 80, podemos concluir que se $p \neq q$, então $\mathtt{(}^p$ não é equivalente a $\mathtt{(}^q$ segundo $\mathcal{R}_L$, pois $\mathtt{(}^p(\mathtt{ab})\,s^p \in L$ mas $\mathtt{(}^q(\mathtt{ab})\,s^p \notin L$, por desacerto do número de parentesis. Portanto, $\mathcal{R}_L$ não é de índice finito e, consequentemente, L não regular. □

Do mesmo modo, a linguagem das palavras que representam *expressões aritméticas* que podem ter parentesis, não é regular. Os autómatos finitos não têm capacidade para verificar emparelhamento de parentesis. No próximo capítulo, estudaremos os autómatos de pilha e veremos que têm capacidade para reconhecer esse tipo de linguagens.

Exemplo 83 A linguagem $L = \{\mathtt{0}^p \mid p \text{ primo}\}$ de alfabeto $\{\mathtt{0}, \mathtt{1}\}$ não é regular.

Já mostrámos este resultado anteriormente. Por isso, sabemos que o conjunto de classes de equivalência de $\mathcal{R}_L$ não pode ser finito.

Não é difícil mostrar que não existem palavras distintas em $\{\varepsilon, \mathtt{0}, \mathtt{0}^2, \mathtt{0}^3, \mathtt{0}^4, \mathtt{0}^5, \mathtt{0}^6, \ldots, \mathtt{0}^{12}, \mathtt{0}^{13}\}$ que sejam equivalentes segundo $\mathcal{R}_L$, pelo que podemos concluir que $\mathcal{R}_L$ tem pelo menos catorze classes de equivalência. Este exemplo pode-nos levar a suspeitar que cada palavra de $\{\mathtt{0}\}^\star$ só seja equivalente (segundo $\mathcal{R}_L$) a si mesma. No entanto, se o nosso conhecimento sobre números primos não ultrapassar a definição de primo, dificilmente vamos conseguir mostrar que se $p \neq q$ então existe z tal que $\mathtt{0}^p z \in L$ e $\mathtt{0}^q z \notin L$. No entanto, sabemos que se $\mathcal{R}_L$ fosse de índice finito, então as classes de $\mathcal{R}_L$ podiam ser vistas como os estados do AFD mínimo que reconhecia L. Assim, podemos fazer uma prova por redução ao absurdo, em tudo semelhante à que fizemos na Proposição 20.

Suponhamos que o conjunto das classes de $\mathcal{R}_L$ é finito e seja n o seu número. Como o conjunto de primos é infinito, seja M um primo tal que $M > 2n$. Como M é primo, $\mathtt{0}^M \in L$.

Existem exatamente $n+1$ palavras em $\{\mathtt{0}\}^\star$ com comprimento menor ou igual a n. Como estamos a supor que $\mathcal{R}_L$ tem apenas n classes, podemos dizer que existem naturais r e s tais que $r \neq s$, $r \leq n$, $s \leq n$ e $[\mathtt{0}^r] = [\mathtt{0}^s]$. Sem perda de generalidade podemos supor que $s > r$. Como $M > 2n$ e $s \leq n$, podemos decompor $\mathtt{0}^M$ como

$$\mathtt{0}^M = \mathtt{0}^r\mathtt{0}^{s-r}\mathtt{0}^{M-s}.$$

Sendo $[\mathtt{0}^r] = [\mathtt{0}^s]$, tem-se[1] $\mathtt{0}^{r+k(s-r)} \in [\mathtt{0}^r]$ para todo $k \geq 1$. Tal resulta da invariância à direita de $\mathcal{R}_L$, pois se $[\mathtt{0}^r] = [\mathtt{0}^s]$ então $[\mathtt{0}^r\mathtt{0}^{s-r}] = [\mathtt{0}^s\mathtt{0}^{s-r}]$. Mas $\mathtt{0}^r\mathtt{0}^{s-r} = \mathtt{0}^s$ e $[\mathtt{0}^r] = [\mathtt{0}^s]$. Analogamente se verifica que se

[1] Convém relembrar que $[\mathtt{0}^r] = [\mathtt{0}^s]$ pode ser interpretado como "$\mathtt{0}^r$ e $\mathtt{0}^s$ levariam o AFD ao mesmo estado". Por outro lado, se compararmos com a prova da Proposição 20, convém notar que $s - r$ pode ser identificado com j.

$[0^{r+k(s-r)}] = [0^r]$ para um certo k fixo (mas qualquer), então $[0^{r+k(s-r)}0^{s-r}] = [0^r 0^{s-r}] = [0^s] = [0^r]$. Ou seja, que se $0^{r+k(s-r)} \in [0^r]$ então $0^{r+(k+1)(s-r)} \in [0^r]$. Conclui-se por indução matemática que

$$[0^r] = [0^s] \;\Rightarrow\; \forall k \in \mathbb{N} \quad [0^{r+k(s-r)}] = [0^r].$$

Se tomarmos $k = M - (s-r)$, temos $[0^s] = [0^{r+(M-(s-r))(s-r)}]$. E, por $\mathcal{R}_L$ ser invariante à direita, $[0^s 0^{M-s}] = [0^{r+(M-(s-r))(s-r)}0^{M-s}]$.

Mas, $r + (M-(s-r))(s-r) + M - s = (M-(s-r))(s-r) + (M-(s-r)) = (M-(s-r))(s-r+1)$. Como $s - r \leq n$ e $M > 2n$ então $M - (s-r) \geq 2$, e sendo $s > r$, também $s - r + 1 \geq 2$. Em conclusão, a palavra $0^{r+(M-(s-r))(s-r)}0^{M-s} \notin L$ pois o seu comprimento não é primo. Mas, $0^s 0^{M-s} = 0^M$ e supusemos que M era primo. Então, as classes $[0^s 0^{M-s}]$ e $[0^{r+(M-(s-r))(s-r)}0^{M-s}]$ têm que ser disjuntas, pois $0^s 0^{M-s}\varepsilon \in L$ e $0^{r+(M-(s-r))(s-r)}0^{M-s}\varepsilon \notin L$ (ou seja, estas palavras não são equivalentes segundo $\mathcal{R}_L$).

A contradição resultou de se ter suposto que o conjunto das classes de equivalência de $\mathcal{R}_L$ era finito. Logo, tal conjunto é infinito. □

Capítulo 5

Autómatos de Pilha

5.1 Noção de autómato de pilha (AP)

Um autómato de pilha é um modelo de uma máquina que tem um **conjunto de estados finito** mas **memória infinita**, suportada por uma *pilha*, com acesso LIFO ("last-in first-out"). Iremos mostrar que estas máquinas são computacionalmente mais potentes do que os autómatos finitos mas, no Capítulo 7, estudaremos um outro tipo de máquinas que são mais potentes do que estas. Embora os autómatos de pilha tenham memória infinita, a sua capacidade é limitada pelo modo como se restringe o acesso à informação colocada na pilha.

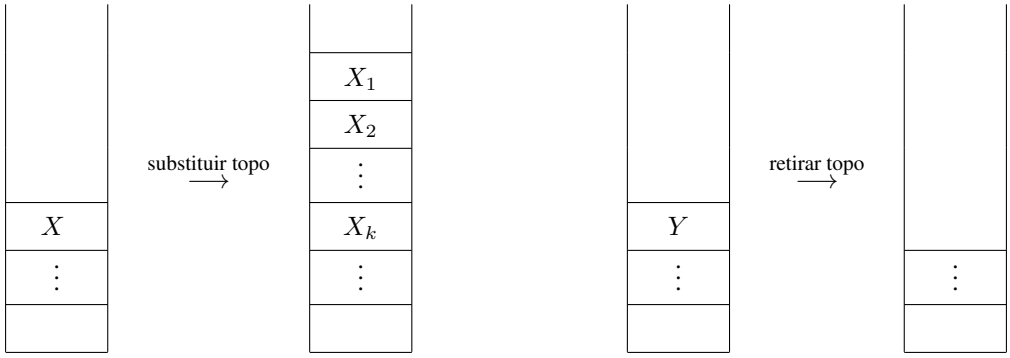
Em cada transição, o autómato pode **alterar o estado e o topo da pilha**. Cada transição é determinada pelo estado atual, pelo símbolo no topo da pilha e possivelmente pelo símbolo da palavra que está a ler. Pode ter também transições por ε. O autómato analisa *o elemento no topo da pilha* e substitui o elemento no topo da pilha por uma sequência de outros elementos (possivelmente vazia). Pode manter inalterado o topo da pilha se efetuar a substituição do topo pelo próprio topo.

Definição 6 *Um autómato de pilha $\mathcal{A}$ é definido por $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, F)$, onde S é o* **conjunto de estados** *(finito), Σ é o* **alfabeto de entrada**, *Γ é o* **alfabeto da pilha**, *$s_0 \in S$ é o* **estado inicial**, *$Z_0 \in \Gamma$ o* **símbolo inicial** *na pilha, F o conjunto de* **estados finais**, *e a função de transição δ é uma função de $S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ em $2^{S \times \Gamma^\star}$.*

Usamos a notação $A \times B \times C$ para referir o conjunto dos ternos ordenados (a, b, c), com $a \in A$, $b \in B$ e $c \in C$, ou seja, $A \times B \times C = \{(a, b, c) \mid a \in A, b \in B, c \in C\}$.

O topo da pilha no início. Quando o autómato começa a processar a palavra, o único símbolo que está na pilha é o símbolo inicial da pilha (acima denotado por Z_0).

Substituição do topo da pilha. Substituir o topo da pilha, X, pela sequência $X_1 \ldots X_k$, com $k \geq 1$, e $X_i \in \Gamma$, corresponde a retirar X e colocar sucessivamente $X_k, \ldots, X_1$, passando o topo a ser X_1. Convenciona-se que *o símbolo mais à esquerda é o que fica no topo.* Substituir o elemento Y que está no topo da pilha por ε corresponde a retirar o topo da pilha.



Critérios de aceitação. Existem dois critérios para reconhecimento de palavras – *aceitação por estados finais* ou *aceitação por pilha vazia*, sendo necessário indicar qual é usado num dado autómato.

- Uma palavra $x \in \Sigma^\star$ é **aceite** pelo autómato de pilha $\mathcal{A}$ **por estados finais** sse x leva $\mathcal{A}$ do estado s_0 a algum dos estados finais, sendo totalmente processada. A linguagem aceite por $\mathcal{A}$ por estados finais é o conjunto das palavras aceites por estados finais.

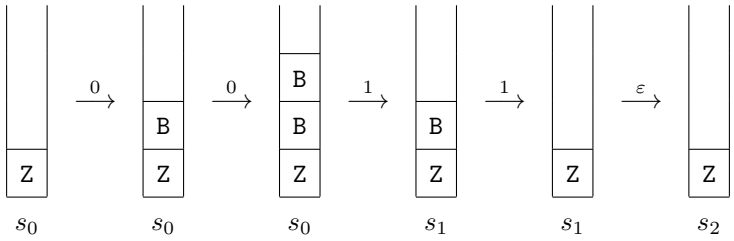
- Uma palavra $x \in \Sigma^\star$ é **aceite** pelo autómato de pilha $\mathcal{A}$ **por pilha vazia** sse x leva $\mathcal{A}$ do estado s_0 a *pilha vazia*, sendo totalmente processada. A linguagem aceite por $\mathcal{A}$ por pilha vazia é o conjunto das palavras aceites por pilha vazia. Diz-se que a **pilha está vazia** quando não tem qualquer símbolo.

Exemplo 84 Considerar o autómato de pilha $\mathcal{A} = (\{s_0, s_1, s_2\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{Z}, \mathtt{B}\}, \delta, s_0, \mathtt{Z}, \{s_2\})$, com:

$$\begin{array}{lcllcl}
\delta(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_2, \mathtt{Z})\} & \delta(s_0, \mathtt{1}, \mathtt{B}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_0, \mathtt{0}, \mathtt{Z}) & = & \{(s_0, \mathtt{BZ})\} & \delta(s_1, \mathtt{1}, \mathtt{B}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_0, \mathtt{0}, \mathtt{B}) & = & \{(s_0, \mathtt{BB})\} & \delta(s_1, \varepsilon, \mathtt{Z}) & = & \{(s_2, \mathtt{Z})\}
\end{array}$$

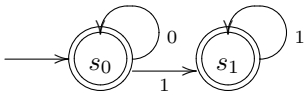
e $\delta(s, \alpha, X) = \{\ \}$, para os restantes ternos $(s, \alpha, X) \in S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$. Em s_0, o autómato $\mathcal{A}$ carrega a pilha e, em s_1, descarrega-a. Como não tem transições que retirem o símbolo inicial $\mathtt{Z}$, a pilha nunca ficará vazia. Assim, a linguagem aceite por $\mathcal{A}$ por estados finais é $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$ e a linguagem aceite por $\mathcal{A}$ por pilha vazia é $\emptyset$.

A título de exemplo, vamos ilustrar esquematicamente o processamento da palavra `0011`.



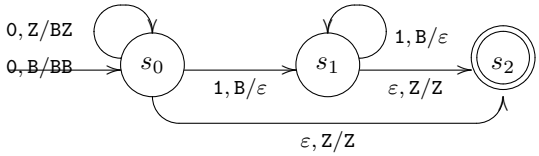
O autómato carrega um B na pilha por cada 0 que encontra. Só pode consumir 1's se tiver B's na pilha e, por cada 1, retira um B da pilha. No estado s_0, consome 0's e mantém-se em s_0. Quando consome o primeiro 1, muda de estado, para evitar o consumo de 0's depois de 1's.

Se compararmos com um autómato finito que reconhece $\{0^n 1^m \mid n, m \in \mathbb{N}\} = \mathcal{L}(0^\star 1^\star)$, vemos que também neste se evita a ocorrência de 0's depois de 1's por uma mudança de estado. Por exemplo:



O autómato de pilha consegue simular a contagem dos 0's, porque os copia para a pilha, a qual, sendo infinita, pode guardar tantos B's quantos forem necessários.

Diagrama de transição para autómato de pilha. Os autómatos de pilha podem ser representados por diagramas como os que introduzimos para autómatos finitos. Se $(s', \gamma) \in \delta(s, a, X)$, o diagrama terá um arco (s, s') com etiqueta $a, X/\gamma$, para $a \in \Sigma \cup \{\varepsilon\}$. Assim, o autómato de pilha definido no Exemplo 84 seria representado pelo diagrama:

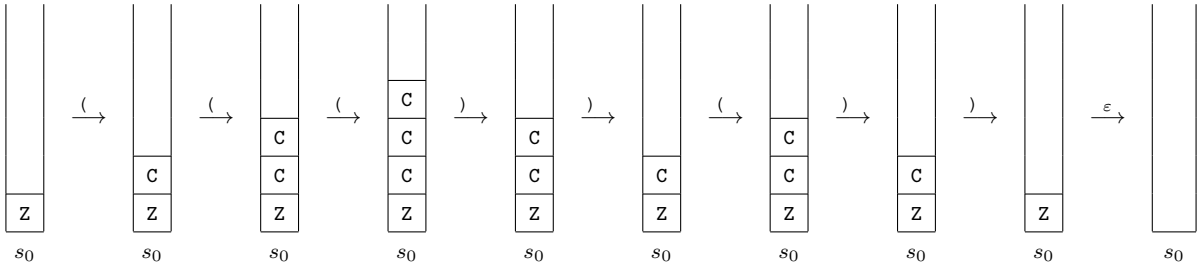


Exemplo 85 Seja $\mathcal{A} = (\{s_0\}, \{\,(\,,\,)\,\}, \{\mathtt{Z}, \mathtt{C}\}, \delta, s_0, \mathtt{Z}, \{\,\})$ um autómato de pilha, com aceitação por pilha vazia, e δ dada por:

$$\begin{aligned}
\delta(s_0, \varepsilon, \mathtt{Z}) &= \{(s_0, \varepsilon)\} \\
\delta(s_0, \mathtt{(}, \mathtt{Z}) &= \{(s_0, \mathtt{CZ})\} \\
\delta(s_0, \mathtt{(}, \mathtt{C}) &= \{(s_0, \mathtt{CC})\} \\
\delta(s_0, \mathtt{)}, \mathtt{C}) &= \{(s_0, \varepsilon)\}
\end{aligned}$$

A linguagem L reconhecida por $\mathcal{A}$ (por pilha vazia) é constituída pela palavra ε e pelas sequências finitas de parentesis curvos que são "bem formadas", isto é, por aquelas cujo número de `(`'s é igual ao número de `)`'s, e que em qualquer seu prefixo o número de `)`'s não excede o número de `(`'s. Tal linguagem é o menor subconjunto de $\{\texttt{(},\texttt{)}\}^\star$ que satisfaz as condições: (i) $\varepsilon \in L$; e (ii) $\texttt{(}\alpha\texttt{)} \in L$ e $\alpha\beta \in L$, quaisquer que sejam $\alpha, \beta \in L$.

O autómato não muda de estado quando detecta parentesis fechados, mas retira um `C` da pilha (o que é correto pois esse `C` representa o seu par). Deste modo, pode continuar a aceitar parentesis abertos. Vamos ilustrar esquematicamente o processamento da palavra `((())())`.



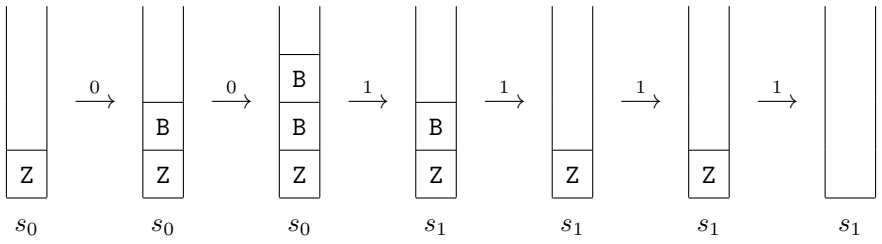
Exemplo 86 Consideremos o autómato de pilha $\mathcal{A} = (\{s_0, s_1\}, \{\texttt{0}, \texttt{1}\}, \{\texttt{Z}, \texttt{B}\}, \delta, s_0, \texttt{Z}, \{\ \})$ onde

$$\begin{array}{lcllcl}
\delta(s_0, \varepsilon, \texttt{Z}) & = & \{(s_1, \texttt{Z})\} & \delta(s_0, \texttt{1}, \texttt{B}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_0, \texttt{0}, \texttt{Z}) & = & \{(s_0, \texttt{BZ})\} & \delta(s_1, \texttt{1}, \texttt{B}) & = & \{(s_1, \varepsilon)\} \\
\delta(s_0, \texttt{0}, \texttt{B}) & = & \{(s_0, \texttt{BB})\} & \delta(s_1, \texttt{1}, \texttt{Z}) & = & \{(s_1, \varepsilon), (s_1, \texttt{Z})\}
\end{array}$$

A linguagem L reconhecida pelo autómato por pilha vazia é

$$L = \{\texttt{0}^n\texttt{1}^m \mid m, n \in \mathbb{N}, m > n\}.$$

Consideremos, por exemplo, um possível processamento de `001111`.



Para concluir que L é a linguagem indicada, observemos que $\mathcal{A}$ só pode consumir `0`'s se estiver em s_0. Depois de consumir $\texttt{0}^n$, com $n \geq 1$, o autómato está no estado s_0 e tem $\texttt{B}^n\texttt{Z}$ na pilha. Em seguida, se consumir k `1`'s, com $1 \leq k \leq n$, fica no estado s_1 e tem $\texttt{B}^{n-k}\texttt{Z}$ na pilha. Quando $k = n$, o autómato está no estado s_1 e tem apenas `Z` na pilha. Estando no estado s_1 e tendo `Z` no topo da pilha, consegue retirar `Z` se consumir qualquer sequência de `1`'s que não seja ε. Notamos ainda que, estando no estado s_0 e tendo `Z` no topo da pilha, $\mathcal{A}$ pode passar a s_1 sem consumir qualquer símbolo, o que permite reconhecer sequências em $\mathcal{L}(\texttt{11}^\star)$. Este autómato *não é determinístico*.

Definição 7 *Um autómato de pilha $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, F)$ é* **determinístico** *se e só se para cada configuração só existir uma transição possível. Ou seja, se e só se, quaisquer que sejam $s \in S$, $a \in \Sigma$ e $X \in \Gamma$, se tem:*

- *se $\delta(s, \varepsilon, X) \neq \emptyset$ então $\delta(s, a, X) = \emptyset$, para todo $a \in \Sigma$,*
- *se $\delta(s, a, X) \neq \emptyset$ então $\delta(s, a, X)$ tem um e um só elemento, quaisquer que sejam $s \in S$, $a \in \Sigma$ e $X \in \Gamma$.*

Convém notar que, ao considerar que alguns autómatos de pilha com transições por ε são *determinísticos*, não estamos a ser coerentes com a terminologia que usámos para autómatos finitos. Recordamos que considerámos que os autómatos finitos com transições por ε eram, por definição, não-determinísticos. O modo como definiamos a função de transição δ ditava tal designação. Aqui, referimo-nos ao *comportamento do autómato*, caraterizando-o como *determinista* ou *não determinista*.

Formalização da noção de configuração. Os esquemas que apresentámos acima podem servir para ilustrar a análise de uma palavra por um autómato mas é útil definir a noção de configuração e de mudança de configuração formalmente. Interessa representar o estado em que o autómato se encontra, o que falta ver da palavra dada e o conteúdo da pilha. Para representar uma **configuração**, usamos ternos (s, x, γ), com $s \in S$, $x \in \Sigma^\star$ e $\gamma \in \Gamma^\star$. À semelhança do que fizemos para autómatos finitos, a **mudança de configuração por uma transição** vai ser traduzida por uma relação binária $\vdash_{\mathcal{A}}$, definida no conjunto de ternos $S \times \Sigma^\star \times \Gamma^\star$ por

$$(s, ax, Z\gamma) \quad \vdash_{\mathcal{A}} \quad (q, x, \beta\gamma) \text{ se e só se } (q, \beta) \in \delta(s, a, Z)$$

para $s \in S$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, $x \in \Sigma^\star$ e $\gamma \in \Gamma^\star$. A **mudança de configuração por n transições** é traduzida pela relação $\vdash^n_{\mathcal{A}}$ (sendo, $\vdash^1_{\mathcal{A}} = \vdash_{\mathcal{A}}$). A **mudança de configuração após um zero ou mais transições** é traduzida por $\vdash^\star_{\mathcal{A}}$, ou seja, pelo fecho reflexivo e transitivo de $\vdash_{\mathcal{A}}$.

Exemplo 87 No exemplo anterior, para descrever a análise da palavra `001111`, podiamos ter escrito:
$(s_0, \texttt{001111}, \texttt{Z}) \vdash_{\mathcal{A}} (s_0, \texttt{01111}, \texttt{BZ}) \vdash_{\mathcal{A}} (s_0, \texttt{1111}, \texttt{BBZ}) \vdash_{\mathcal{A}} (s_1, \texttt{111}, \texttt{BZ}) \vdash_{\mathcal{A}} (s_1, \texttt{11}, \texttt{Z}) \vdash_{\mathcal{A}} (s_1, \texttt{1}, \texttt{Z}) \vdash_{\mathcal{A}} (s_1, \varepsilon, \varepsilon)$.
Portanto, $(s_0, \texttt{001111}, \texttt{Z}) \vdash^6_{\mathcal{A}} (s_1, \varepsilon, \varepsilon)$ e, consequentemente, $(s_0, \texttt{001111}, \texttt{Z}) \vdash^\star_{\mathcal{A}} (s_1, \varepsilon, \varepsilon)$. Logo, `001111` é aceite por $\mathcal{A}$ por pilha vazia.

Definição 8 *As linguagens aceites pelo autómato de pilha $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, F)$ por pilha vazia e por estados finais são $\{x \mid \exists s \in S\ (s_0, x, Z_0) \vdash^\star_{\mathcal{A}} (s, \varepsilon, \varepsilon)\}$ e $\{x \mid \exists f \in F\ \exists \gamma \in \Gamma^\star\ (s_0, x, Z_0) \vdash^\star_{\mathcal{A}} (f, \varepsilon, \gamma)\}$, respetivamente.*

Para cada autómato, terá de se indicar o critério de aceitação a aplicar, sendo usual definir $F = \emptyset$ se o critério for "aceitação por pilha vazia". Pela Definição 8, em ambos os casos, exige-se que a palavra tenha sido toda lida, uma vez

que, na configuração final, a palavra que falta ver é ε. Caso o critério de aceitação seja pilha vazia, então o conteúdo da pilha deve ser ε, não interessando o estado s em que o autómato se encontra. Se o critério for aceitação por estado final, então o conteúdo γ da pilha não interessa, mas exige-se que o estado f seja um estado final de $\mathcal{A}$.

Questões de notação... Sendo $\vdash_{\mathcal{A}}$ uma relação binária, podiamos escrever $(c_1, c_2) \in \vdash_{\mathcal{A}}$. Mas, neste contexto, é preferível usar a notação infixa, ou seja, $c_1 \vdash_{\mathcal{A}} c_2$. Para facilitar a notação, escrevemos, $c_1 \vdash_{\mathcal{A}} c_2 \vdash_{\mathcal{A}} c_3 \vdash_{\mathcal{A}} c_4$ como abreviatura de $(c_1 \vdash_{\mathcal{A}} c_2) \wedge (c_2 \vdash_{\mathcal{A}} c_3) \wedge (c_3 \vdash_{\mathcal{A}} c_4)$. Sempre que estivermos a considerar um único autómato de pilha, podemos escrever $c_1 \vdash c_2 \vdash c_3 \vdash c_4$.

5.2 Algumas propriedades e algoritmos de conversão

Proposição 28 *A classe de linguagens aceites por autómatos de pilha por estados finais é a classe de linguagens aceites por autómatos de pilha por pilha vazia.*

Ideia da prova: Seja L uma qualquer linguagem de alfabeto Σ.

- Vamos mostrar que, se L é aceite por um autómato de pilha $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, F)$ por estados finais então é aceite por algum autómato de pilha por pilha vazia. A ideia é completar o autómato $\mathcal{A}$ de forma que, sempre que chegar a estado final, possa esvaziar a pilha sem consumir qualquer símbolo da palavra. Para isso, consideramos o autómato

$$\mathcal{A}' = (S \cup \{s_0', q\}, \Sigma, \Gamma \cup \{Z_0'\}, \delta', s_0', Z_0', \emptyset)$$

em que Z_0' é um novo símbolo (ou seja, $Z_0' \notin \Gamma$), s_0' e q são novos estados, sendo q o estado que $\mathcal{A}'$ vais usar para esvaziar a pilha, e δ' dada por:

$$\begin{aligned}
\delta'(s_0', \varepsilon, Z_0') &= \{(s_0, Z_0 Z_0')\} \\
\delta'(s, a, Z) &= \delta(s, a, Z), \text{para todo } s \in S, a \in \Sigma, Z \in \Gamma \\
\delta'(s, \varepsilon, Z) &= \delta(s, \varepsilon, Z), \text{para todo } s \in S \setminus F, Z \in \Gamma \\
\delta'(f, \varepsilon, Z) &= \delta(f, \varepsilon, Z) \cup \{(q, \varepsilon)\}, \text{para todo } f \in F, Z \in \Gamma \\
\delta'(f, \varepsilon, Z_0') &= \{(q, \varepsilon)\}, \text{para todo } f \in F \\
\delta'(s, \varepsilon, Z_0') &= \{\ \}, \text{para todo } s \in S \setminus F \\
\delta'(q, \varepsilon, Z) &= \{(q, \varepsilon)\}, \text{para todo } Z \in \Gamma \cup \{Z_0'\}
\end{aligned}$$

Podia acontecer que uma dada palavra x levasse autómato $\mathcal{A}$ a um estado não final mas a pilha vazia. Para evitar que x passasse agora a ser aceite por $\mathcal{A}'$, introduziu-se um novo símbolo inicial de pilha (representado por Z_0'). Pode-se provar que $\mathcal{A}'$ aceita L por pilha vazia (mas omitiremos essa parte da prova).

- Para provar que se L é aceite por pilha vazia por um autómato de pilha $\mathcal{A} = (S, \Sigma, \Gamma, \delta, s_0, Z_0, \emptyset)$, então L é aceite por algum autómato de pilha $\mathcal{A}'$ por estados finais, a ideia é acrescentar transições para que o autómato passe a estado final sempre que a pilha estiver vazia. Uma das possibilidades seria identificar as transições que retiram Z_0 e substitui-las por transições para estado final. No entanto, se Z_0 tiver sido colocado na pilha várias vezes, esta ideia pode não funcionar. Para evitar isso, vamos considerar um novo símbolo inicial Z_0', tal que $Z_0' \notin \Gamma$, e dois novos estados f e s_0'. O autómato $\mathcal{A}'$, que vamos definir, imita $\mathcal{A}$ mas, quando $\mathcal{A}$ ficar com a pilha vazia, $\mathcal{A}'$ ainda vai ter Z_0' na pilha, passando nessa altura a estado final. O autómato $\mathcal{A}'$ pode ser definido por $\mathcal{A}' = (S \cup \{f, s_0'\}\}, \Sigma, \Gamma \cup \{Z_0'\}, \delta', s_0', Z_0', \{f\})$, onde,

$$\begin{aligned}
\delta'(s_0', \varepsilon, Z_0) &= \{(s_0, Z_0 Z_0')\} \\
\delta'(s_0', a, Z_0) &= \{\ \}, \text{ para todo } a \in \Sigma \\
\delta'(s, a, Z) &= \delta(s, a, Z) \text{ para todo } s \in S, a \in \Sigma \text{ e } Z \in \Gamma \\
\delta'(s, \varepsilon, Z_0') &= \{(f, Z_0')\}, \text{ para todo } s \in S \\
\delta'(s, a, Z_0') &= \{\ \}, \text{ para todo } s \in S \cup \{f\} \text{ e } a \in \Sigma
\end{aligned}$$

  Pode-se provar que $\mathcal{A}'$ aceita L por estados finais (mas omiteremos essa parte da prova). □

Exemplo 88 Por conversão do autómato apresentado no Exemplo 86, vamos definir um autómato de pilha $\mathcal{A}'$ que reconheça $\{\mathtt{0}^n\mathtt{1}^m \mid m, n \in \mathbb{N},\ m > n\}$ por estados finais, Seguindo o método de conversão descrito na prova da Proposição 28, inserimos um novo símbolo inicial, denotado por `U` e dois novos estados q_0 e f, e construimos $\mathcal{A}' = (\{s_0, q_0, f\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{U}, \mathtt{Z}, \mathtt{B}\}, \delta', q_0, \mathtt{U}, \{f\})$, com δ' dada por

$$\begin{aligned}
\delta'(s_0, \varepsilon, \mathtt{Z}) &= \{(s_1, \mathtt{Z})\} & \qquad \delta'(s_0, \mathtt{1}, \mathtt{B}) &= \{(s_1, \varepsilon)\} \\
\delta'(s_0, \mathtt{0}, \mathtt{Z}) &= \{(s_0, \mathtt{BZ})\} & \qquad \delta'(s_1, \mathtt{1}, \mathtt{B}) &= \{(s_1, \varepsilon)\} \\
\delta'(s_0, \mathtt{0}, \mathtt{B}) &= \{(s_0, \mathtt{BB})\} & \qquad \delta'(s_1, \mathtt{1}, \mathtt{Z}) &= \{(s_1, \varepsilon), (s_1, \mathtt{Z})\} \\
\delta'(q_0, \varepsilon, \mathtt{U}) &= \{(s_0, \mathtt{ZU})\} \\
\delta'(s_0, \varepsilon, \mathtt{U}) &= \{(f, \mathtt{U})\} \\
\delta'(s_1, \varepsilon, \mathtt{U}) &= \{(f, \mathtt{U})\}
\end{aligned}$$

sendo $\delta'(s, a, X) = \emptyset$, para os restantes ternos em $\{s_0, q_0, f\} \times \{\varepsilon, \mathtt{0}, \mathtt{1}\} \times \{\mathtt{U}, \mathtt{Z}, \mathtt{B}\}$.

Como exemplo, consideramos a análise da palavra `001111`: $(q_0, \mathtt{001111}, \mathtt{U}) \vdash (s_0, \mathtt{001111}, \mathtt{ZU}) \vdash (s_0, \mathtt{01111}, \mathtt{BZU}) \vdash (s_0, \mathtt{1111}, \mathtt{BBZU}) \vdash (s_1, \mathtt{111}, \mathtt{BZU}) \vdash (s_1, \mathtt{11}, \mathtt{ZU}) \vdash (s_1, \mathtt{1}, \mathtt{ZU}) \vdash (s_1, \varepsilon, \mathtt{U}) \vdash (f, \varepsilon, \mathtt{U})$.

Exemplo 89 Por conversão do autómato de pilha apresentado no Exemplo 84, vamos definir um autómato de pilha $\mathcal{A}'$ que reconhece $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$ por pilha vazia.

Seguindo o método de conversão descrito, seja $\mathcal{A}' = (\{s_0, s_1, s_2, s'_0, q\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{Z}, \mathtt{B}, \mathtt{U}\}, \delta', s'_0, \mathtt{U}, \{\ \})$, com

$$\begin{array}{lcllcl}
\delta'(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_2, \mathtt{Z})\} & \delta'(s_0, \mathtt{1}, \mathtt{B}) & = & \{(s_1, \varepsilon)\} \\
\delta'(s_0, \mathtt{0}, \mathtt{Z}) & = & \{(s_0, \mathtt{BZ})\} & \delta'(s_1, \mathtt{1}, \mathtt{B}) & = & \{(s_1, \varepsilon)\} \\
\delta'(s_0, \mathtt{0}, \mathtt{B}) & = & \{(s_0, \mathtt{BB})\} & \delta'(s_1, \varepsilon, \mathtt{Z}) & = & \{(s_2, \mathtt{Z})\} \\
\delta'(s'_0, \varepsilon, \mathtt{U}) & = & \{(s_0, \mathtt{ZU})\} & \delta'(s_2, \varepsilon, \mathtt{Z}) & = & \{(q, \varepsilon)\} \\
\delta'(s_2, \varepsilon, \mathtt{B}) & = & \{(q, \varepsilon)\} & \delta'(s_2, \varepsilon, \mathtt{U}) & = & \{(q, \varepsilon)\} \\
\delta'(q, \varepsilon, \mathtt{B}) & = & \{(q, \varepsilon)\} & \delta'(q, \varepsilon, \mathtt{Z}) & = & \{(q, \varepsilon)\} \\
\delta'(q, \varepsilon, \mathtt{U}) & = & \{(q, \varepsilon)\} & & &
\end{array}$$

e $\delta(s, \alpha, X) = \{\ \}$, para os restantes ternos $(s, \alpha, X) \in \{s_0, s_1, s_2, s'_0, q\} \times \{\mathtt{0}, \mathtt{1}, \varepsilon\} \times \{\mathtt{Z}, \mathtt{B}, \mathtt{U}\}$.

Neste exemplo, bem como no anterior, pretendemos ilustrar os métodos de conversão descritos na prova da Proposição 28. Contudo, facilmente se reconhece que existe um autómato de pilha que é mais simples do que $\mathcal{A}'$ e que aceita L por pilha vazia. Bastaria $\mathcal{A}'' = (\{s_0, s_1\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{Z}, \mathtt{B}\}, \delta'', \mathtt{Z}, \{\ \})$ com δ'' dada por:

$$\begin{array}{lcllcl}
\delta''(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_1, \varepsilon)\} & \delta''(s_0, \mathtt{1}, \mathtt{B}) & = & \{(s_1, \varepsilon)\} \\
\delta''(s_0, \mathtt{0}, \mathtt{Z}) & = & \{(s_0, \mathtt{BZ})\} & \delta''(s_1, \mathtt{1}, \mathtt{B}) & = & \{(s_1, \varepsilon)\} \\
\delta''(s_0, \mathtt{0}, \mathtt{B}) & = & \{(s_0, \mathtt{BB})\} & \delta''(s_1, \varepsilon, \mathtt{Z}) & = & \{(s_1, \varepsilon)\}
\end{array}$$

e $\delta(s, \alpha, X) = \{\ \}$, para os restantes ternos $(s, \alpha, X) \in \{s_0, s_1\} \times \{\mathtt{0}, \mathtt{1}, \varepsilon\} \times \{\mathtt{Z}, \mathtt{B}\}$. Para que a definição da função de transição esteja completa é necessário incluir esta última referência, embora, por vezes, seja omitida, ficando apenas implicíto $\delta(s, \alpha, X) = \{\ \}$, para todos os ternos não mencionados. A transição $\delta''(s_1, \varepsilon, \mathtt{Z}) = \{(s_1, \varepsilon)$ seria desnecessária se, em vez de $\delta''(s_0, \mathtt{0}, \mathtt{Z}) = \{(s_0, \mathtt{BZ})\}$, tivessemos definido $\delta''(s_0, \mathtt{0}, \mathtt{Z})$ por $\{(s_0, \mathtt{B})\}$.

Proposição 29 *A classe de linguagens aceites por autómatos de pilha não determinísticos contém propriamente a classe de linguagens aceites por autómatos de pilha determinísticos.*

Ideia da prova: Se o alfabeto Σ tiver pelo menos dois símbolos, existem linguagens de alfabeto Σ que não são reconhecidas por qualquer autómato de pilha determinístico, mas que podem ser reconhecidas por autómatos de pilha não determinísticos. Por exemplo, a linguagem das capícuas de comprimento par (Exemplo 90).

Se uma linguagem for aceite por um autómato de pilha determinístico, também pode ser aceite por um não determinístico. A ideia é simples. Basta introduzir mais uma transição *sem qualquer interesse* num dos conjuntos $\delta(s, a, Z)$ não vazios (por exemplo, uma transição para um novo estado do qual não há mais transições). Desta forma, obtém-se um autómato não determinístico que reconhece a mesma linguagem que o autómato dado. Observemos que, a ideia pode ser adaptada também se não $\mathcal{A}$ não tiver transições. □

Para a Proposição 29, é necessário que o alfabeto Σ tenha pelo menos dois símbolos, pois é conhecido que *para* $|\Sigma| = 1$*, qualquer linguagem sobre* Σ *que seja aceite por um autómato de pilha de pilha é regular* (o que, como veremos, implica que, para alfabetos com apenas um símbolo, a classe de linguagens regulares coincide com a classe de linguagens independentes de contexto). A prova deste resultado pode ser encontrada na bibliografia.

Exemplo 90 A linguagem $L = \{ww^r \mid w \in \{\mathtt{a},\mathtt{b}\}^\star\}$, onde w^r denota a sequência w escrita da direita para a esquerda, é aceite por pilha vazia pelo autómato de pilha $\mathcal{A} = (\{s_0, s_1\}, \{\mathtt{a},\mathtt{b}\}, \{\mathtt{Z},\mathtt{A},\mathtt{B}\}, \delta, s_0, \mathtt{Z}, \{\ \})$ onde

$$
\begin{array}{lcllcl}
\delta(s_0,\varepsilon,\mathtt{Z}) & = & \{(s_1,\mathtt{Z})\} & & & \\
\delta(s_0,\mathtt{a},\mathtt{Z}) & = & \{(s_0,\mathtt{AZ})\} & \delta(s_0,\mathtt{b},\mathtt{Z}) & = & \{(s_0,\mathtt{BZ})\} \\
\delta(s_0,\mathtt{a},\mathtt{B}) & = & \{(s_0,\mathtt{AB})\} & \delta(s_0,\mathtt{b},\mathtt{A}) & = & \{(s_0,\mathtt{BA})\} \\
\delta(s_0,\mathtt{a},\mathtt{A}) & = & \{(s_0,\mathtt{AA}),(s_1,\varepsilon)\} & \delta(s_0,\mathtt{b},\mathtt{B}) & = & \{(s_0,\mathtt{BB}),(s_1,\varepsilon)\} \\
\delta(s_1,\mathtt{a},\mathtt{A}) & = & \{(s_1,\varepsilon)\} & \delta(s_1,\mathtt{b},\mathtt{B}) & = & \{(s_1,\varepsilon)\} \\
\delta(s_1,\varepsilon,\mathtt{Z}) & = & \{(s_1,\varepsilon)\} & & &
\end{array}
$$

O não determinismo permite ter transições no estado s_0 para passagem ao estado s_1, para começar a descarregar a pilha (imaginando que já consumiu metade da palavra dada) e para se manter no estado s_0, continuando a carregar a pilha (imaginando que ainda não consumiu metade da palavra dada). No estado s_1, retirará da pilha cada `A` com um `a` e cada `B` com `b`. Como o autómato não pode saber exatamente se a palavra está a meio, tem que ser necessariamente não determinístico.

Pela Definição 8, **apenas interessa encontrar um "percurso" de sucesso, se se quiser mostrar que uma dada palavra é aceite pelo autómato**. Assim, por exemplo, `abaaba` é aceite por $\mathcal{A}$ por pilha vazia, uma vez que $(s_0, \mathtt{abaaba}, \mathtt{Z}) \vdash (s_0, \mathtt{baaba}, \mathtt{AZ}) \vdash (s_0, \mathtt{aaba}, \mathtt{BAZ}) \vdash (s_0, \mathtt{aba}, \mathtt{ABAZ}) \vdash (s_1, \mathtt{ba}, \mathtt{BAZ}) \vdash (s_1, \mathtt{a}, \mathtt{AZ}) \vdash (s_1, \varepsilon, \mathtt{Z}) \vdash (s_1, \varepsilon, \varepsilon)$ e, portanto, $(s_0, \mathtt{abaaba}, \mathtt{Z}) \vdash^\star (s_1, \varepsilon, \varepsilon)$.

Ou seja, o que importa para decidir que $\mathtt{abaaba} \in L$ é o facto de $(s_0, \mathtt{abaaba}, \mathtt{Z}) \vdash^\star (s_1, \varepsilon, \varepsilon)$, ainda que se tenha também, por exemplo, $(s_0, \mathtt{abaaba}, \mathtt{Z}) \vdash^\star (s_0, \varepsilon, \mathtt{ABAABAZ})$.

De modo análogo, podemos definir um autómato de pilha não determinístico para reconhecer a linguagem das capícuas de qualquer comprimento, par ou ímpar, isto é, $\{wtw^r \mid w \in \Sigma^\star, t \in \Sigma \cup \{\varepsilon\}\}$, com $\Sigma = \{\mathtt{a},\mathtt{b}\}$. Esta linguagem não pode ser reconhecida por um autómato de pilha determinístico.

Exemplo 91 A linguagem $L = \{w\mathtt{c}w^r \mid w \in \{\mathtt{a},\mathtt{b}^\star\}$ de alfabeto $\{\mathtt{a},\mathtt{b},\mathtt{c}\}$ pode ser aceite por um autómato de pilha determinístico, pois o símbolo `c` marca o meio da palavra. Por exemplo, $\mathcal{A} = (\{s_0, s_1\}, \{\mathtt{a},\mathtt{b},\mathtt{c}\}, \{\mathtt{Z},\mathtt{A},\mathtt{B}\}, \delta, s_0, \mathtt{Z}, \{\ \})$ onde δ é dada por:

$$
\begin{array}{lcl@{\qquad}lcl}
\delta(s_0,\mathtt{c},\mathtt{Z}) & = & \{(s_0,\varepsilon)\} & & & \\
\delta(s_0,\mathtt{a},\mathtt{Z}) & = & \{(s_0,\mathtt{AZ})\} & \delta(s_0,\mathtt{b},\mathtt{Z}) & = & \{(s_0,\mathtt{BZ})\} \\
\delta(s_0,\mathtt{a},\mathtt{B}) & = & \{(s_0,\mathtt{AB})\} & \delta(s_0,\mathtt{b},\mathtt{A}) & = & \{(s_0,\mathtt{BA})\} \\
\delta(s_0,\mathtt{a},\mathtt{A}) & = & \{(s_0,\mathtt{AA})\} & \delta(s_0,\mathtt{b},\mathtt{B}) & = & \{(s_0,\mathtt{BB})\} \\
\delta(s_0,\mathtt{c},\mathtt{A}) & = & \{(s_1,\mathtt{A})\} & \delta(s_0,\mathtt{c},\mathtt{B}) & = & \{(s_1,\mathtt{B})\} \\
\delta(s_1,\mathtt{a},\mathtt{A}) & = & \{(s_1,\varepsilon)\} & \delta(s_1,\mathtt{b},\mathtt{B}) & = & \{(s_1,\varepsilon)\} \\
\delta(s_1,\varepsilon,\mathtt{Z}) & = & \{(s_1,\varepsilon)\} & & &
\end{array}
$$

Proposição 30 *A classe de linguagens aceites por autómatos finitos está contida (propriamente) na classe de linguagens aceites por autómatos de pilha.*

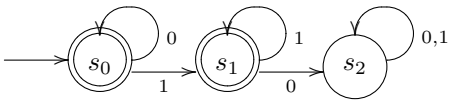
Ideia da prova: Se L for aceite por um autómato finito, então existe algum autómato de pilha que aceita L. Sabemos que se L é aceite por algum autómato finito, então é aceite por algum AFD. Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um tal autómato. É simples ver que o autómato de pilha $\mathcal{A}_P = (S, \Sigma, \{\mathtt{Z}\}, \delta', s_0, \mathtt{Z}, F)$, com $\delta'(s, a, \mathtt{Z}) = \{(\delta(s, a), \mathtt{Z})\}$, aceita L por estados finais. O autómato $\mathcal{A}'$ "imita" o autómato finito $\mathcal{A}$ e não efetua qualquer alteração da pilha durante a análise das palavras.

A inclusão é estrita se Σ tem pelo menos dois elementos (mas as classes são iguais se Σ só tiver um símbolo, como referimos anteriormente). Já vimos exemplos de linguagens que não são regulares e são aceites por autómatos de pilha (por exemplo, Exemplos 84, 85, 86, 90 91). □

Corolário 30.1 *A classe das linguagens regulares está contida na classe das linguagens reconhecidas por autómatos de pilha determinísticos com aceitação por estados finais.*

Prova: Segue da prova da Proposição 30, sendo verdade que, se $|\Sigma| \geq 2$, então a inclusão é estrita. □

Exemplo 92 Para o AFD representado obtém-se o autómato de pilha $\mathcal{A}' = \{s_0, s_1, s_2\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{Z}\}, \delta', \mathtt{s_0}, \mathtt{Z}, \{\mathtt{s_0}, \mathtt{s_1}\})$, com aceitação por estados finais, estando δ' definida abaixo à direita.



$$
\begin{array}{ll}
\delta'(s_0,\mathtt{0},\mathtt{Z}) = \{(s_0,\mathtt{Z})\} & \delta'(s_0,\mathtt{1},\mathtt{Z}) = \{(s_1,\mathtt{Z})\} \\
\delta'(s_1,\mathtt{0},\mathtt{Z}) = \{(s_2,\mathtt{Z})\} & \delta'(s_1,\mathtt{1},\mathtt{Z}) = \{(s_1,\mathtt{Z})\} \\
\delta'(s_2,\mathtt{0},\mathtt{Z}) = \{(s_2,\mathtt{Z})\} & \delta'(s_2,\mathtt{1},\mathtt{Z}) = \{(s_2,\mathtt{Z})\}
\end{array}
$$

Capítulo 6

Linguagens independentes de contexto e Autómatos de Pilha

Neste capítulo vamos introduzir a noção de linguagem independente de contexto e provar que a classe de linguagens aceites por autómatos de pilha é a classe das linguagens independentes de contexto.

6.1 Linguagens independentes de contexto (LICs)

Definição 9 *Uma* **gramática independente de contexto (GIC)** *é um quarteto* $\mathcal{G} = (V, \Sigma, P, S)$*, onde* V *e* Σ *são conjuntos de símbolos, tais que* $V \cap \Sigma = \emptyset$*, sendo ambos finitos e não vazios,* $S \in V$*, e* P *é uma relação binária de* V *em* $(V \cup \Sigma)^\star$*, sendo* $|P|$ *finito, sendo usadas as designações seguintes:*

- V *é o conjunto das* **variáveis** *ou símbolos* **não terminais***;*
- S *é* **símbolo inicial***;*
- Σ *é o* **alfabeto** *ou conjunto dos símbolos* **terminais***;*
- P *é o conjunto de* **produções** *ou* **regras** *e, usualmente, escreve-se* $X \to w$ *se* $(X, w) \in P$*. A qualquer uma das regras de produção que definem* X*, com* $X \in V$*, chama-se* X**-produção**.

Os símbolos não terminais caraterizam *categorias gramaticais*, sendo S o que identifica a categoria principal, ou seja, a que determina a linguagem que a gramática gera. Estas gramáticas chamam-se "independentes de contexto" porque as regras que definem os não-terminais são aplicadas sem estarem sujeitas a restrições de contexto. Mais

formalmente, estas gramáticas são caraterizadas por terem um único símbolo do lado esquerdo de cada regra, sendo da forma $X \to w$, onde X é necessariamente uma variável da gramática.

A forma das regras de produção determina a expressividade das gramáticas. Nas gramáticas mais gerais (ditas *Tipo 0*), as regras têm a forma $\alpha \to \beta$ com $\alpha, \beta \in (V \cup \Sigma)^\star$, $\alpha \neq \varepsilon$. Mas, não iremos estudar gramáticas desse tipo.

Exemplo 93 Seja L a linguagem de alfabeto $\{\mathtt{a}, \mathtt{b}\}$ das palavras obtidas por aplicação das regras (r_1) e (r_2), uma ou mais vezes:

(r_1) $\quad \mathtt{aaa} \in L$

(r_2) $\quad \alpha\mathtt{bb}\beta \in L$, quaisquer que sejam $\alpha, \beta \in L$.

Podemos representar (r_1) e (r_2) pelas duas regras seguintes:

$$\begin{aligned} S &\to \mathtt{aaa} \\ S &\to S\mathtt{bb}S \end{aligned}$$

as quais definem a categoria gramatical S, dizendo que uma palavra da categoria S é $\mathtt{aaa}$ ou a justaposição de três sequências: uma qualquer palavra do tipo S, $\mathtt{bb}$ e uma qualquer do tipo S.

Alternativamente, como $L = \mathcal{L}((\mathtt{aaabb})^\star\mathtt{aaa})$, podemos ainda definir L como sendo a linguagem gerada por aplicação das regras:

$$\begin{aligned} S &\to \mathtt{aaa} \\ S &\to \mathtt{aaabb}S \end{aligned}$$

Exemplo 94 Seja $L_1 = \{\mathtt{0}^{2k}\mathtt{1}^k \mid k \in \mathbb{N}\}$. A linguagem L_1 é o menor subconjunto de $\{\mathtt{0}, \mathtt{1}\}^\star$ que satisfaz as duas condições (i) e (ii) seguintes.

(i) $\quad \varepsilon \in L_1$

(ii) $\quad \mathtt{00}x\mathtt{1} \in L_1$, para todo $x \in L_1$

Equivalentemente, podemos dizer que L_1 é constituída por todas as palavras que possam ser formadas por aplicação, uma ou mais vezes, das regras (r_1) e (r_2) seguintes

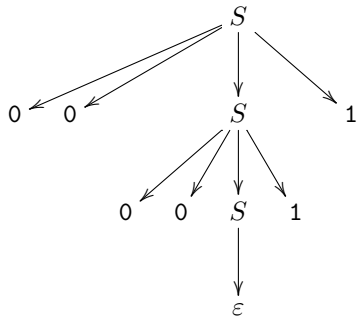
(r_1) $\quad \varepsilon \in L_1$

(r_2) $\quad$ Qualquer que seja x, se $x \in L_1$ então $\mathtt{00}x\mathtt{1} \in L_1$

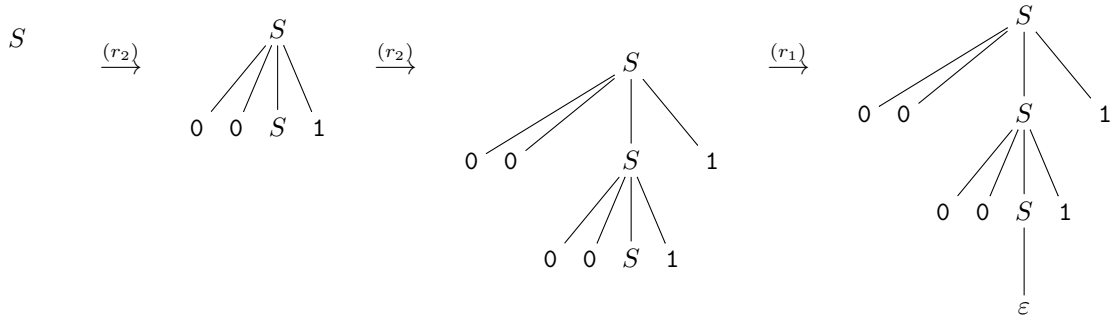
Por analogia com o exemplo anterior, caraterizar as palavras de L_1 como sendo da categoria S dada por:

$$\begin{aligned} S &\to \varepsilon \\ S &\to \mathtt{00}S\mathtt{1} \end{aligned}$$

A palavra `000011` é do tipo S, como se ilustra pelo seguinte esquema, a que se chama *árvore de derivação (ou árvore sintáctica)* para a palavra `000011`.



Esta árvore foi construída do modo seguinte (para simplificar representamos segmentos em vez de setas):



Exemplo 95 Seja F a linguagem de alfabeto $\Sigma = \{\texttt{f}, \texttt{x}, \texttt{(}, \texttt{)}, \texttt{,}\}$ assim definida indutivamente.

(i) $\texttt{x} \in F$

(ii) $\texttt{f(}\alpha_1\texttt{,}\alpha_2\texttt{)} \in F$, quaisquer que sejam $\alpha_1,\ \alpha_2 \in F$

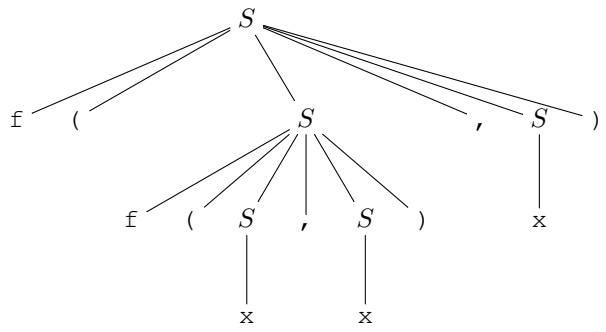
Alguns exemplos de palavras desta linguagem:

$$\texttt{f(x,x)}$$
$$\texttt{f(f(x,x),x)}$$
$$\texttt{f(x,f(x,x))}$$
$$\texttt{f(f(x,x),f(x,x))}$$
$$\texttt{f(f(x,x),f(f(x,x),x))}$$

Considerando que as palavras da linguagem F são da categoria S, podemos escrever as regras seguintes:

$$\begin{aligned} S &\rightarrow \texttt{x} \\ S &\rightarrow \texttt{f(}S\texttt{,}S\texttt{)} \end{aligned}$$

A árvore sintáctica de $\mathtt{f(f(x,x),x)}$ é:



Exemplo 96 Seja $L_2 = \{\mathtt{0}^{2k}\mathtt{1}^k \mid k \geq 0\} \cup \{\mathtt{1}^k\mathtt{0}^{2k} \mid k \geq 0\}$, que é definida indutivamente pelas regras (i)—(iii).

(i) $\varepsilon \in L_2$

(ii) se $x \in L_2$ e $x \notin \{\mathtt{0},\mathtt{1}\}^*\{\mathtt{1}\}$ então $\mathtt{1}x\mathtt{00} \in L_2$

(iii) se $x \in L_2$ e $x \notin \{\mathtt{0},\mathtt{1}\}^*\{\mathtt{0}\}$ então $\mathtt{00}x\mathtt{1} \in L_2$

Se observarmos L_2, vemos que é constituída por dois tipos de palavras: as da forma $\mathtt{0}^{2k}\mathtt{1}^k$, que passamos a referir como sendo da categoria A e as da forma $\mathtt{1}^k\mathtt{0}^{2k}$, que passamos a referir como sendo da categoria B. As palavras de L_2 são do tipo A ou do tipo B, e designamos a sua categoria gramatical por S. As regras da gramática são:

$$\begin{array}{lcl} S & \to & A \\ S & \to & B \end{array} \qquad \begin{array}{lcl} A & \to & \varepsilon \\ A & \to & \mathtt{00}\,A\,\mathtt{1} \end{array} \qquad \begin{array}{lcl} B & \to & \varepsilon \\ B & \to & \mathtt{1}\,B\,\mathtt{00} \end{array}$$

Notamos que cada palavra de L_2, com excepção da palavra ε, tem uma só árvore sintáctica.

A palavra ε tem duas árvores sintácticas:

$$\begin{array}{c} S \\ | \\ A \\ | \\ \varepsilon \end{array} \qquad \text{ou} \qquad \begin{array}{c} S \\ | \\ B \\ | \\ \varepsilon \end{array}$$

Por esta razão, a gramática diz-se *ambígua*. Se as regras forem as indicadas abaixo, então a gramática já não seria ambígua. Para cada palavra da linguagem existiria uma e uma só árvore sintáctica. A gramática fixaria uma estrutura sintáctica única para cada palavra.

$$\begin{array}{lcl} S & \to & \varepsilon \\ S & \to & \mathtt{00}\,A\,\mathtt{1} \\ S & \to & \mathtt{1}\,B\,\mathtt{00} \end{array} \qquad \begin{array}{lcl} A & \to & \varepsilon \\ A & \to & \mathtt{00}\,A\,\mathtt{1} \end{array} \qquad \begin{array}{lcl} B & \to & \varepsilon \\ B & \to & \mathtt{1}\,B\,\mathtt{00} \end{array}$$

Exemplo 97 A linguagem $L_3 = \mathcal{L}(\ \mathtt{0(0+1)^\star 1}\ ) = \{\mathtt{0}w\mathtt{1} \mid w \in \{\mathtt{0},\mathtt{1}\}^\star\}$ é gerada pela gramática seguinte, em que S é o símbolo inicial. A regra para S define o padrão $\mathtt{0}w\mathtt{1}$, com $w \in \{\mathtt{0},\mathtt{1}\}^\star$, pois T gera $\{\mathtt{0},\mathtt{1}\}^\star$.

$$\begin{aligned} S &\rightarrow \mathtt{0}\,T\,\mathtt{1} \\ T &\rightarrow \mathtt{0}\,T \\ T &\rightarrow \mathtt{1}\,T \\ T &\rightarrow \varepsilon \end{aligned}$$

Por vezes, vamos escrever as mesmas regras de uma forma abreviada, como se ilustra a seguir (onde "|" se lê "ou"):

$$\begin{aligned} S &\rightarrow \mathtt{0}\,T\,\mathtt{1} \\ T &\rightarrow \mathtt{0}\,T \mid \mathtt{1}\,T \mid \varepsilon \end{aligned}$$

Exemplo 98 A linguagem $L = \{\mathtt{0}x_1 \ldots \mathtt{0}x_k\mathtt{0} \mid k \geq 1, x_i \in \mathcal{L}(\ (\mathtt{11})^\star\mathtt{11}\ ) \text{ para } 1 \leq i \leq k\}$ é definida pela seguinte gramática, com símbolo inicial S:

$$\begin{aligned} S &\rightarrow \mathtt{0}\,X\,R \\ R &\rightarrow \mathtt{0}\,X\,R \mid \mathtt{0} \\ X &\rightarrow \mathtt{11}\,X \mid \mathtt{11} \end{aligned}$$

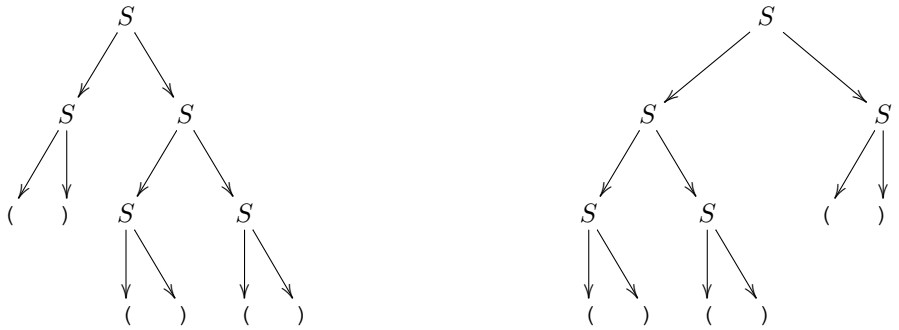
Verificamos que:

- X gera $\mathcal{L}(\ (\mathtt{11})^\star\mathtt{11}\ )$
- R gera $\mathcal{L}(\ (\mathtt{0}(\mathtt{11})^\star\mathtt{11})^\star\mathtt{0}\ )$
- S gera $L = \mathcal{L}(\ (\mathtt{0}(\mathtt{11})^\star\mathtt{11})^\star\mathtt{0}(\mathtt{11})^\star\mathtt{110}\ )$

Exemplo 99 As sequências de parentesis curvos que são "bem formadas", no sentido de terem igual número de `(`'s e `)`'s, e serem tais que em qualquer prefixo de uma tal sequência o número de `)`'s não excede o número de `(`'s, é a linguagem de alfabeto $\{\,\mathtt{(},\mathtt{)}\,\}$ gerada pelas regras seguintes:

$$S \rightarrow \mathtt{()} \mid SS \mid \mathtt{(}\,S\,\mathtt{)}$$

A palavra `()()()` tem duas árvores de derivação.



Esta ambiguidade resulta de as regras não fixarem o modo como uma sequência de S's deve ser partida. Assim se se tiver SSS (como no caso de `()()()`), podemos considerar que é justaposição de SS com S, mas também podemos considerar que é justaposição de S com SS. Se tivessemos escrito

$$\begin{array}{lcl} S & \to & X \mid XS \\ X & \to & \texttt{(}S\texttt{)} \mid \texttt{()} \end{array}$$

definiamos a mesma linguagem, mas agora cada palavra teria uma só arvore sintáctica (ou seja, a gramática não seria ambígua). Embora a justificação cuidadosa deste facto não seja trivial, não é difícil concluir que X não pode ser uma sequência de S's.

Exemplo 100 Este exemplo sugere uma motivação para o estudo das gramáticas no âmbito do curso — a descrição de linguagens de programação. Que palavras de $\{\texttt{(}, \texttt{)}, \texttt{:}, \texttt{>}, \texttt{<}, =, +, -, *, /, \texttt{0}, \ldots, \texttt{9}, \texttt{a}, \texttt{b}, \texttt{c}, \ldots, \texttt{z}, \cdot, \texttt{;}\}^\star$ são geradas a partir de S por aplicação das regras seguintes?

$$\begin{array}{lcl} S & \to & \texttt{if}\cdot\texttt{(}C\texttt{)}\cdot\texttt{then}\cdot\texttt{goto}\cdot N \\ S & \to & \texttt{goto}\cdot N \\ S & \to & V\texttt{:=}E \\ S & \to & \texttt{stop} \\ E & \to & \texttt{(}E\,O\,E\texttt{)} \mid \texttt{(}-E\texttt{)} \mid V \mid N \\ O & \to & + \mid - \mid * \mid / \\ C & \to & V\,X\,E \\ X & \to & \texttt{>} \mid \texttt{<} \mid \texttt{>=} \mid \texttt{<=} \mid \texttt{=} \mid \texttt{<>} \\ N & \to & D \mid D\,N \\ D & \to & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\ V & \to & L \mid L\,N \mid L\,V \\ L & \to & \texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \texttt{d} \mid \texttt{e} \mid \texttt{f} \mid \texttt{g} \mid \texttt{h} \end{array}$$

E, se definirmos como símbolo inicial a variável P, acrescentando as regras seguintes?

$$P \to N\cdot S \mid S\,\texttt{;}\cdot\cdot\, P$$

Exemplo duma palavra da linguagem gerada a partir de P:

$$\texttt{2}\cdot\texttt{a:=10;}\cdot\cdot\,\texttt{3}\cdot\texttt{b:=35;}\cdot\cdot\,\texttt{4}\cdot\texttt{if}\cdot\texttt{(b}<\texttt{a)}\cdot\texttt{then}\cdot\texttt{goto}\cdot\texttt{8;}\cdot\cdot\,\texttt{6}\cdot\texttt{b:=(b}-\texttt{a);}\cdot\cdot\,\texttt{7}\cdot\texttt{goto}\cdot\texttt{4;}\cdot\cdot\,\texttt{8}\cdot\texttt{stop}$$

Se supusermos que $\cdot\cdot$ representa o caracter "a mudança de linha" e $\cdot$ o caracter "espaço", então a palavra anterior seria:

```
2 a:=10;
3 b:=35;
4 if (b<a) then goto 8;
6 b:=(b-a);
7 goto 4;
8 stop
```

Exemplo 101 Que palavras de alfabeto $\{\texttt{(},\texttt{)},+,^\star,\emptyset,\boxed{\varepsilon},\texttt{a},\texttt{b}\}$ são geradas por aplicação das regras seguintes?

$$\begin{array}{rcl} S & \rightarrow & \texttt{a} \mid \texttt{b} \mid \emptyset \mid \boxed{\varepsilon} \\ S & \rightarrow & \texttt{(}S{+}S\texttt{)} \mid \texttt{(}SS\texttt{)} \mid \texttt{(}S^\star\texttt{)} \end{array}$$

Se admitirmos que $\boxed{\varepsilon}$ está a representar o símbolo ε, podemos dizer que esta gramática descreve a linguagem das expressões regulares de alfabeto $\{\texttt{a},\texttt{b}\}$. Saliente-se que se tivessemos escrito $S \rightarrow \varepsilon$, a gramática não geraria, por exemplo, a expressão $\texttt{(a+}\varepsilon\texttt{)}$, mas geraria $\texttt{(a+)}$, palavra que não é uma expressão regular! Para evitar esta confusão, foi necessário distinguir o símbolo ε que pode ocorrer nas expressões regulares, e por isso usámos $\boxed{\varepsilon}$. Assim, esta gramática gera $\texttt{(a+}\boxed{\varepsilon}\texttt{)}$.

Não é demais salientar que, as palavras desta linguagem **são expressões regulares**. A linguagem que constituem não é regular.

6.1.1 Linguagem gerada por uma gramática e ambiguidade

A linguagem que uma dada gramática gera é constituída pelas palavras que esta caracteriza como sendo da categoria correspondente ao seu símbolo inicial. Por isso, para bem caracterizar uma gramática, é imprescindível indicar qual é o seu **símbolo inicial**.

Exemplo 102 Podemos verificar que cada uma das gramáticas gera a linguagem indicada à sua direita:

$$\begin{array}{rcll} \mathcal{G}_1 & = & (\{T\},\{\texttt{0},\texttt{1}\},\{T \rightarrow \texttt{0}T,\ T \rightarrow \texttt{1}T,\ T \rightarrow \varepsilon\},T) & \mathcal{L}(\mathcal{G}_1) = \{\texttt{0},\texttt{1}\}^\star \\ \mathcal{G}_2 & = & (\{Z,U\},\{\texttt{0},\texttt{1}\},\{Z \rightarrow \texttt{0}U,\ U \rightarrow \texttt{1}Z,\ U \rightarrow \varepsilon\},Z) & \mathcal{L}(\mathcal{G}_2) = \{\texttt{0}\}\{\texttt{10}\}^\star = \mathcal{L}(\mathcal{G}_5) \\ \mathcal{G}_3 & = & (\{Z,U\},\{\texttt{0},\texttt{1}\},\{Z \rightarrow \texttt{0}U,\ U \rightarrow \texttt{1}Z,\ U \rightarrow \varepsilon\},U) & \mathcal{L}(\mathcal{G}_3) = \{\texttt{10}\}^\star = \mathcal{L}(\mathcal{G}_4) \\ \mathcal{G}_4 & = & (\{U\},\{\texttt{0},\texttt{1}\},\{U \rightarrow \texttt{10}U,\ U \rightarrow \varepsilon\},U) & \\ \mathcal{G}_5 & = & (\{U,Z\},\{\texttt{0},\texttt{1}\},\{Z \rightarrow \texttt{0}U,\ U \rightarrow \texttt{10}U,\ U \rightarrow \varepsilon\},Z) & \end{array}$$

$$\mathcal{G}_6 = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, \{S \to \mathtt{aaa},\ S \to \mathtt{aaabb}S\}, S) \quad \mathcal{L}(\mathcal{G}_6) = \{\mathtt{aaabb}\}^\star\{\mathtt{aaa}\}$$

$$\mathcal{G}_7 = (\{S, C\}, \{\mathtt{a}, \mathtt{b}\}, \{S \to \mathtt{a}S\mathtt{b},\ S \to C,\ C \to \mathtt{b}C\mathtt{a},\ C \to \mathtt{b}\}, S) \quad \mathcal{L}(\mathcal{G}_7) = \{\mathtt{a}^n\mathtt{b}^{m+1}\mathtt{a}^m\mathtt{b}^n \mid m, n \in \mathbb{N}\}$$

Para formalizar a noção de **linguagem gerada por uma gramática** começamos por definir **derivação**.

Definição 10 *Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma gramática independente de contexto. Para $x, y \in (V \cup \Sigma)^\star$, diz-se que x* **deriva imediatamente** *y se e só se $x = x_1 X x_2$, $y = x_1\, w\, x_2$, e $(X \to w) \in P$, com $x_1, x_2, w \in (V \cup \Sigma)^\star$ e $X \in V$. Ou seja, $X \to w$ é a produção que se aplica para de x se derivar y, por substituição de X em x por w. A relação de derivação imediata (ou num passo) é denotada por $\Rightarrow_{\mathcal{G}}$ (ou por $\Rightarrow$, se não houver outra gramática em causa), escrevendo-se $x \Rightarrow_{\mathcal{G}} y$.*

Questões de notação... O símbolo $\Rightarrow$ denota também uma das conectivas lógicas (abreviatura de "**se**...**então**"). Assim, para evitar confusão, é preferível não abreviar $\Rightarrow_{\mathcal{G}}$ por $\Rightarrow$, mesmo que haja uma única gramática envolvida.

Do ponto de vista formal, $\Rightarrow_{\mathcal{G}}$ é uma relação binária em $(V \cup \Sigma)^\star$, a que podemos chamar **relação de derivação imediata**. Esta relação determina ainda outras relações binárias em $(V \cup \Sigma)^\star$:

$\Rightarrow_{\mathcal{G}}^n$ **relação de derivação em n passos**, $n \in \mathbb{N}$. Corresponde a efetuar sucessivamente exatamente n derivações imediatas, onde n pode ser qualquer $n \geq 0$. Tem-se $\Rightarrow_{\mathcal{G}}^1 = \Rightarrow_{\mathcal{G}}$ (derivação num passo é a derivação imediata), $\Rightarrow_{\mathcal{G}}^2$ (derivação em dois passos).... Tem-se, $x \Rightarrow_{\mathcal{G}}^2 y$ sse $x \Rightarrow_{\mathcal{G}} z$ e $z \Rightarrow_{\mathcal{G}} y$ para algum $z \in (V \cup \Sigma)^\star$. Convencionamos que $\Rightarrow_{\mathcal{G}}^0$ é a relação identidade em $(V \cup \Sigma)^\star$, a qual é dada por $\{(x, x) \mid x \in (V \cup \Sigma)^\star\}$.

Ou seja, $x \Rightarrow_{\mathcal{G}}^0 x$, para todo $x \in (V \cup \Sigma)^\star$.

Podemos definir por recorrência $\Rightarrow_{\mathcal{G}}^n$, para $n \geq 1$, do modo seguinte:

$$\begin{aligned} \Rightarrow_{\mathcal{G}}^1 &= \Rightarrow_{\mathcal{G}} \\ \Rightarrow_{\mathcal{G}}^{k+1} &= \{(x, y) \mid \exists z \in (V \cup \Sigma)^\star \ (x \Rightarrow_{\mathcal{G}}^k z \ \wedge\ z \Rightarrow_{\mathcal{G}} y)\}, \text{ com } k \geq 1 \end{aligned}$$

Ou seja, tem-se $x \Rightarrow_{\mathcal{G}}^{k+1} y$ se e só se existe $z \in (V \cup \Sigma)^\star$ tal que $(x \Rightarrow_{\mathcal{G}}^k z \ \wedge\ z \Rightarrow_{\mathcal{G}} y)$.

Usando a definição de composição de relações binárias, podemos mostrar que

$$x \Rightarrow_{\mathcal{G}}^k y \quad \text{se e só se} \quad \exists i \in \mathbb{N}\, \exists z \in (V \cup \Sigma)^\star \ (i \leq k \ \wedge\ x \Rightarrow_{\mathcal{G}}^i z \ \wedge\ z \Rightarrow_{\mathcal{G}}^{k-i} y)$$

o que traduz o facto de qualquer derivação em k passos poder ser vista como uma *composição* de uma derivação em i passos com uma derivação em $k - i$ passos, com $0 \leq i \leq k$.

$\Rightarrow_{\mathcal{G}}^{\star}$ **relação de derivação**: traduz derivações por aplicação das regras de derivação zero ou mais vezes. Relembrando que as relações binárias são conjuntos (de pares ordenados dum dado conjunto), a relação $\Rightarrow_{\mathcal{G}}^{\star}$ não é mais do que a união das $\Rightarrow_{\mathcal{G}}^{n}$, com $n \geq 0$, ou seja $\Rightarrow_{\mathcal{G}}^{\star} = \bigcup_{n\geq 0}(\Rightarrow_{\mathcal{G}}^{n})$. De facto, é o fecho reflexivo e transitivo da relação $\Rightarrow_{\mathcal{G}}$, ou seja, $\Rightarrow_{\mathcal{G}}^{\star}$ é a menor relação binária em $(V \cup \Sigma)^{\star}$ que é reflexiva, transitiva e contém $\Rightarrow_{\mathcal{G}}$.

Como, se $x \neq y$ então $x \not\Rightarrow_{\mathcal{G}}^{0} y$, é óbvio que, se $x \Rightarrow_{\mathcal{G}}^{\star} y$ e $x \neq y$ então a derivação terá pelo menos um passo.

Definição 11 *A* **linguagem gerada pela gramática** $\mathcal{G} = (V, \Sigma, P, S)$ *é dada por* $\mathcal{L}(\mathcal{G}) = \{x \mid x \in \Sigma^{\star} \wedge S \Rightarrow_{\mathcal{G}}^{\star} x\}$*, ou seja, é o conjunto das sequências de terminais que se podem derivar a partir do símbolo inicial* S*, por aplicação das regras da gramática. Uma* **linguagem independente de contexto** *é uma linguagem que é gerada por alguma gramática independente de contexto.*

Dada $x \in \mathcal{L}(\mathcal{G})$, uma árvore de derivação de x "dá uma estrutura a x". A definição de árvore de derivação (ou sintáctica) que damos a seguir usa a noção de árvore ordenada.

Definição 12 *A* **árvore de derivação de** $x \in \mathcal{L}(\mathcal{G})$ *é uma árvore ordenada tal que: a raíz é* S*, o símbolo numa folha é um terminal ou* ε*, a palavra* x *é concatenação dos símbolos nas folhas, o símbolo num nó interno é uma variável, e usou-se* $X \to s_1 \ldots s_n$ *sse* X *é nó (interno) e tem filhos* $s_1 \ldots s_n$*.*

Exemplo 103 Consideremos novamente a gramática referida no Exemplo 98, que definimos agora, mais formalmente, por $\mathcal{G} = (\{S, R, X\}, \{\mathtt{0}, \mathtt{1}\}, \mathcal{P}, S)$, com o conjunto das regras $\mathcal{P}$ é consituído por

$$\begin{array}{lcl} S & \to & \mathtt{0}\,X\,R \\ R & \to & \mathtt{0}\,X\,R \mid \mathtt{0} \\ X & \to & \mathtt{11}\,X \mid \mathtt{11} \end{array}$$

Vamos mostrar que esta gramática gera `011110`. Para isso, vamos partir de S (símbolo inicial de $\mathcal{G}$) e aplicar sucessivamente regras da gramática, não esquecendo que pretendemos chegar a `011110`. Em cada passo da derivação, substituimos uma variável qualquer, aplicando uma regra de $\mathcal{G}$. Por exemplo:

$$S \Rightarrow_{\mathcal{G}} \mathtt{0}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{011}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{011}\,X\,\mathtt{0} \Rightarrow_{\mathcal{G}} \mathtt{011110}$$

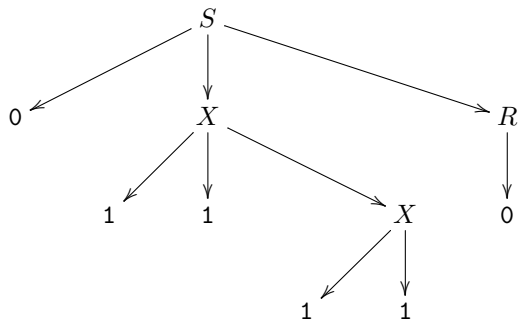
Existem outras derivações para `011110`. Se substituirmos sempre a *variável* mais à esquerda, temos **uma derivação** que dizemos ser **pela esquerda**.

$$S \Rightarrow_{\mathcal{G}} \mathtt{0}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{011}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{01111}\,R \Rightarrow_{\mathcal{G}} \mathtt{011110}$$

Se substituirmos sempre a *variável* mais à direita, temos **uma derivação pela direita**.

$$S \Rightarrow_{\mathcal{G}} \texttt{0}\,X\,R \Rightarrow_{\mathcal{G}} \texttt{0}\,X\,\texttt{0} \Rightarrow_{\mathcal{G}} \texttt{011}\,X\texttt{0} \Rightarrow_{\mathcal{G}} \texttt{011110}$$

Apresentámos derivações diferentes para `011110`. Contudo, todas determinam a mesma árvore de derivação para `011110`. É possível verificar que existe uma única **árvore de derivação** para `011110`.

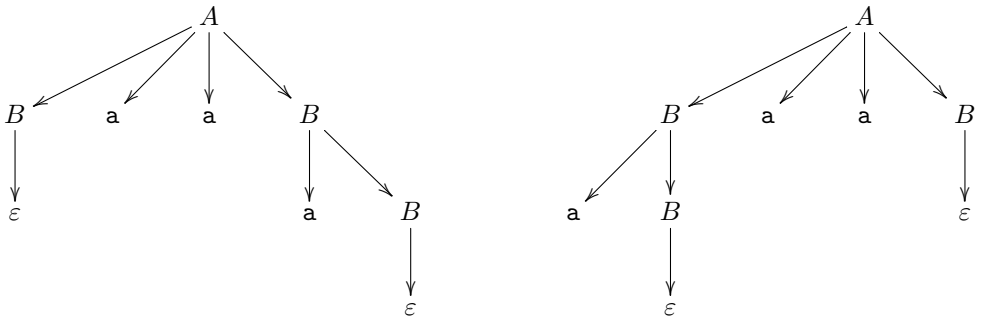


Exemplo 104 Seja $\mathcal{G}$ a gramática $(\{A, B\}, \{\texttt{a}, \texttt{b}\}, \{A \to B\texttt{aa}B,\ B \to \texttt{a}B,\ B \to \texttt{b}B,\ B \to \varepsilon\}, A)$

Duas derivações pela esquerda para `aaa`:

$$A \Rightarrow B\texttt{aa}B \Rightarrow \varepsilon\texttt{aa}B \Rightarrow \texttt{aaa}B \Rightarrow \texttt{aaa}\varepsilon = \texttt{aaa}$$
$$A \Rightarrow B\texttt{aa}B \Rightarrow \texttt{a}B\texttt{aa}B \Rightarrow \texttt{a}\varepsilon\texttt{aa}B \Rightarrow \texttt{aaa}\varepsilon = \texttt{aaa}$$

Duas árvores de derivação para `aaa` determinadas por essas derivações:



Definição 13 *Uma* **gramática** $\mathcal{G}$ *é* **ambígua** *sse se existe mais do que uma árvore de derivação para algum* $x \in \mathcal{L}(\mathcal{G})$*. Como cada árvore de derivação corresponde a uma e uma só derivação pela esquerda (ou pela direita), é equivalente dizer que* $\mathcal{G}$ *é ambígua sse existe* $x \in \mathcal{L}(\mathcal{G})$ *tal que* x *admite mais do que uma derivação pela esquerda (direita).*

Definição 14 *Uma linguagem independente de contexto é uma* **linguagem (inerentemente) ambígua** *se e só se todas as gramáticas independentes de contexto que a geram são ambíguas.*

Exemplo 105 Existem linguagens independentes de contexto que são ambíguas. Podemos encontrar, por exemplo em [Hopcroft & Ullman], a prova de que $\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i,j,k \in \mathbb{N}, \text{ e } i=j \text{ ou } j=k\}$ de alfabeto $\{\mathtt{a},\mathtt{b},\mathtt{c}\}$ é ambígua.

Exemplo 106 A gramática $\mathcal{G} = (\{A,B\},\{\mathtt{a},\mathtt{b}\},\{A \to B\mathtt{aa}B,\ B \to \mathtt{a}B,\ B \to \mathtt{b}B,\ B \to \varepsilon\}, A)$ e a gramática $\mathcal{G}' = (\{A,B,C\},\{\mathtt{a},\mathtt{b}\},P,A)$, onde P é dado por

$$\begin{array}{lcl} A \to C\,\mathtt{aa}\,B & \qquad & \begin{array}{lcl} B \to \mathtt{a}\,B & \qquad & C \to \mathtt{b}\,C \\ B \to \mathtt{b}\,B & & C \to \mathtt{ab}\,C \\ B \to \varepsilon & & C \to \varepsilon \end{array} \end{array}$$

são equivalentes, pois $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}') = \mathcal{L}((\mathtt{a}+\mathtt{b})^\star\mathtt{aa}(\mathtt{a}+\mathtt{b})^\star)$, ou seja, ambas geram a linguagem constituída pelas palavras de alfabeto $\{\mathtt{a},\mathtt{b}\}$ que têm $\mathtt{aa}$ como subpalavra.

Mas, $\mathcal{G}$ é ambígua e $\mathcal{G}'$ não é ambígua. De facto, $\mathcal{G}'$ "identifica" a primeira ocorrência de $\mathtt{aa}$ na palavra, e por isso não há qualquer ambiguidade quanto à estrutura da palavra. A gramática $\mathcal{G}'$ decompõe x em

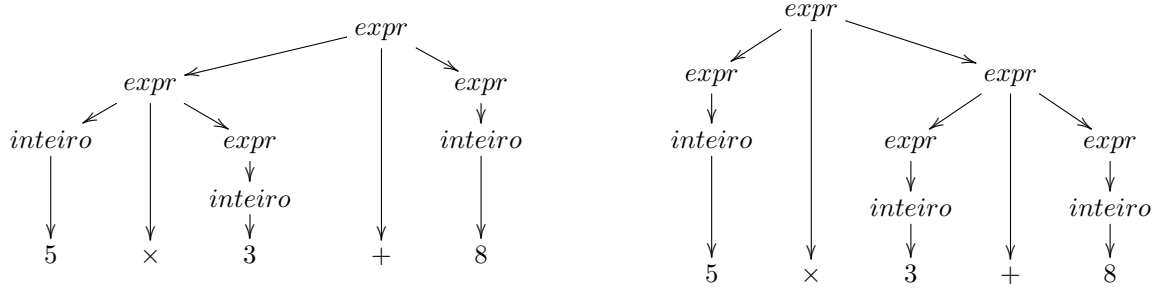
$$x_1\mathtt{aa}x_2 \quad \text{com} \quad x_1 \in \{\mathtt{b},\mathtt{ab}\}^\star,\ \ x_2 \in \{\mathtt{a},\mathtt{b}\}^\star$$

enquanto que $\mathcal{G}$ decompõe em

$$x_1\mathtt{aa}x_2 \quad \text{com} \quad x_1, x_2 \in \{\mathtt{a},\mathtt{b}\}^\star$$

Assim, concluimos que a linguagem $\mathcal{L}((\mathtt{a}+\mathtt{b})^\star\mathtt{aa}(\mathtt{a}+\mathtt{b})^\star)$ não é ambígua. Mais adiante, vamos ver que nenhuma linguagem regular é ambígua.

Exemplo 107 Uma gramática que defina expressões aritméticas não deve ser ambígua. Se, por exemplo, $\mathtt{5} \times \mathtt{3} + \mathtt{8}$ tivesse duas árvores de derivação, poderia ter duas interpretações distintas que, possivelmente, conduziriam a valores distintos se a expressão fosse avaliada seguindo essa estrutura. Também, não pode haver ambiguidade na definição de uma linguagem de programação (se não, a semântica/comportamento dos programas poderia não ser única).



6.1.2 Propriedades de fecho

Vamos provar que a classe das linguagens independentes de contexto é fechada para a união finita, a concatenação e o fecho de Kleene. Na secção 6.5, provamos que não é fechada para a complementação nem para a intersecção.

Proposição 31 *A união finita de linguagens independentes de contexto é uma linguagem independente de contexto.*

Prova: (por indução matemática) Sejam L_1 e L_2 linguagens independentes de contexto (**LICs**) de alfabeto Σ e $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ gramáticas independentes de contexto, com $L_1 = \mathcal{L}(\mathcal{G}_1)$ e $L_2 = \mathcal{L}(\mathcal{G}_2)$. Sem perda de generalidade, vamos supor que $V_1 \cap V_2 = \emptyset$, pois, se $V_1 \cap V_2 \neq \emptyset$, bastaria escolher outros símbolos para renomear as variáveis comuns, evitando assim colisões de nomes.

Seja $G = (V, \Sigma, P, S)$, em que S é uma nova váriavel, $V = \{S\} \cup V_1 \cup V_2$, e $P = \{S \to S_1,\ S \to S_2\} \cup P_1 \cup P_2$. Não é dificil concluir que $L_1 \cup L_2 = \mathcal{L}(G)$. De facto, por definição, $x \in \mathcal{L}(G)$ sse $x \in \Sigma^\star$ e $S \Rightarrow^\star x$. Mas, $S \Rightarrow^\star x$ se e só se $S \Rightarrow S_1$ e $S_1 \Rightarrow^\star x$, ou $S \Rightarrow S_2$ e $S_2 \Rightarrow^\star x$. Portanto, a união de duas LICs é uma LIC.

Como hipótese de indução, vamos supor que, para um dado $n \geq 2$, a união de n quaisquer LICs de alfabeto Σ, é uma LIC. Seja L a união de $n+1$ LICs, ou seja, L é definida por $L = L_1 \cup L_2 \cup \cdots \cup L_n \cup L_{n+1}$. Para concluir que L é uma LIC, basta notar que $L = (L_1 \cup L_2 \cup \cdots \cup L_n) \cup L_{n+1}$. Como, por hipótese, a união de n LICs é uma LIC, conclui-se que L é união de duas LICs, e, portanto, L é uma LIC (como provámos já acima). □

Proposição 32 *O fecho de Kleene duma linguagem independente de contexto é independente de contexto.*

Prova: Seja L uma LIC e seja $G_1 = (V_1, \Sigma, P_1, S_1)$ uma gramática independente de contexto (GIC) que gera L. Consideremos $G = (\{S\} \cup V_1, \Sigma, \{\ S \to \varepsilon,\ \ S \to S_1 S\ \} \cup P_1, S)$, sendo S uma nova variável.
Vamos mostrar que a linguagem $L^\star$ é gerada pela GIC G.

- $\mathcal{L}(G) \subseteq L^\star$.
  Qualquer que seja $n \geq 1$ tem-se $S \Rightarrow^n x$ sem aplicar regras em P_1 se e só se $x = S_1^n S$ ou $x = S_1^{n-1}$. Logo, se $S \Rightarrow^\star x$ e $x \in \Sigma^\star$ então x é uma sequência finita de palavras de L ou $x = \varepsilon$. Portanto, $\mathcal{L}(G) \subseteq L^\star$.
- $\mathcal{L}(G) \supseteq L^\star$.
  Se $x \in L^\star$ então $x = \varepsilon$ ou x é uma sequência de k palavras de L, para algum $k \in \mathbb{N}$. Logo, se for $x = \varepsilon$, é trivial que $S \Rightarrow^\star x$; se for $x = x_1 \ldots x_k$ com $x_i \in L$, então $S \Rightarrow^{k+1} S_1^k \Rightarrow^\star x_1 S_1^{k-1} \Rightarrow^\star x$. □

Proposição 33 *A concatenação de duas LICs é uma LIC.*

Prova: Sejam L_1 e L_2 LICs e sejam $G_1 = (V_1, \Sigma, P_1, S_1)$ e $G_2 = (V_2, \Sigma, P_2, S_2)$ GICs que as geram. Sem perda de generalidade, supomos $V_1 \cap V_2 = \emptyset$. Seja S uma nova variável. Não é difícil concluir que $L_1 L_2$ é a linguagem gerada pela gramática independente de contexto $G = (\{S\} \cup V_1 \cup V_2, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$. □

Exercício 6.1.1 Por indução matemática, mostrar que a concatenação finita de linguagens independentes de contexto é independente de contexto.

6.1.3 Inclusão da classe de linguagens regulares na classe de LICs

Nesta secção, vamos ver que a classe das linguagens regulares está contida propriamente na classe das linguagens independentes de contexto. Tal prova segue quase como corolário do facto de a classe das linguagens independentes de contexto ser fechada para as operações atrás referidas (união, concatenação e fecho de Kleene). Apresentamos ainda uma prova alternativa, da qual resulta um método de conversão de um autómato finito para uma gramática independente de contexto equivalente.

Proposição 34 *Qualquer linguagem regular é independente de contexto.*

Prova: Seja $L \subseteq \Sigma^\star$ uma linguagem regular, e seja r uma expressão regular que descreve L. Vamos mostrar a proposição por indução sobre o número de operadores de r.

- (Caso de base) Se a expressão r tem zero operadores, temos:
  - se $r = \varepsilon$, seja $G = (\{S\}, \Sigma, \{S \to \varepsilon\}, S)$;
  - se $r = \emptyset$, seja $G = (\{S\}, \Sigma, \{\ \}, S)$;
  - se $r = a$ com $a \in \Sigma$, seja $G = (\{S\}, \Sigma, \{S \to a\}, S)$.
- (Hereditariedade) Consideramos o caso em que a expressão r tem $n + 1$ operadores, com $n \geq 0$ fixo. Como hipótese de indução, supomos que qualquer linguagem que seja descrita por uma expressão regular r' com menos operadores do que r, é independente de contexto, sendo r' qualquer.

  Como r tem algum operador, então $r = (r_1^\star)$, ou $r = (r_1 r_2)$ $r = (r_1 + r_2)$, para expressões regulares r_1 e r_2 com menos operadores do que r. Logo, por hipótese, $\mathcal{L}(r_1)$ e $\mathcal{L}(r_2)$ são LICs. Então, como mostrámos que a classe das linguagens independentes de contexto é fechada para a união, o fecho de Kleene e a concatenação, concluimos que $\mathcal{L}(r)$ é uma LIC, pois $\mathcal{L}(r) = (\mathcal{L}(r_1))^\star$, ou $\mathcal{L}(r) = \mathcal{L}(r_1)\mathcal{L}(r_2)$ ou $\mathcal{L}(r) = \mathcal{L}(r_1) \cup \mathcal{L}(r_1)$. □

Conversão de autómatos finitos para GICs. Em alternativa, podemos mostrar a Proposição 34 por construção de uma GIC a partir de um AFD que reconhece a linguagem regular dada. Seja L essa linguagem e $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ um AFD tal que $L = \mathcal{L}(\mathcal{A})$. Podemos mostrar que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G})$, para a gramática $\mathcal{G} = (\mathcal{V}, \Sigma, P, V_{q_0})$ assim definida:

- $\mathcal{V} = \{V_q \mid q \in Q\}$: a cada *estado* $q \in Q$ do autómato associamos uma *variável* V_q.

- O conjunto de regras P é assim definido:
  - A *transição* $\delta(q,a) = q'$ é traduzida pela *regra* $V_q \rightarrow aV_{q'}$
  - Para todo estado final $f \in F$, inclui ainda a regra $V_f \rightarrow \varepsilon$.

$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G})$ pois cada percurso em $\mathcal{A}$ com origem em q_0 (estado inicial do AFD) vai corresponder a exatamente uma derivação em $\mathcal{G}$, com partida de V_{q_0} (símbolo inicial de $\mathcal{G}$). Se recordarmos que a relação de mudança de configuração de $\mathcal{A}$ é denotada por $\vdash_{\mathcal{A}}$, concluimos que, para todo $q, q' \in Q$, $f \in F$, $x \in \Sigma^\star$, $a \in \Sigma$ e $n \in \mathbb{N}$, se tem:

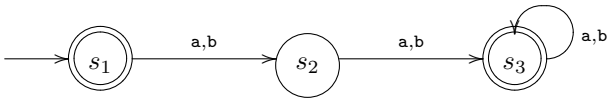
$$\begin{aligned}
(q, ax) \vdash_{\mathcal{A}} (q', x) \quad &\text{sse} \quad V_q \Rightarrow_{\mathcal{G}} aV_{q'} \\
(q, x) \vdash^n_{\mathcal{A}} (q', \varepsilon) \quad &\text{sse} \quad V_q \Rightarrow^n_{\mathcal{G}} xV_{q'} \\
(q_0, x) \vdash^\star_{\mathcal{A}} (f, \varepsilon) \quad &\text{sse} \quad V_{q_0} \Rightarrow^\star_{G} xV_f \Rightarrow_{\mathcal{G}} x
\end{aligned}$$

Observação. O método de conversão que acabámos de expor pode ser aplicado a qualquer autómato finito para obter uma GIC equivalente. Não é necessário que seja um AFD, mas se for um AFD a gramática obtida será não ambígua.

Proposição 35 *Qualquer linguagem regular é não ambígua.*

Prova: Por conversão de um AFD que reconheça a linguagem L dada, usando o método exposto. A GIC que se obtém é não ambígua, porque cada percurso no AFD de q_0 para um estado final f determina uma e uma só derivação, tendo-se $(q_0, x) \vdash^\star_{\mathcal{A}} (f, \varepsilon)$ sse $V_{q_0} \Rightarrow^\star_{G} xV_f \Rightarrow_{\mathcal{G}} x$. □

Exemplo 108 Consideremos o autómato seguinte:



A gramática construída pelo método de conversão que acabámos de expor tem $\{V_{s_1}, V_{s_2}, V_{s_3}\}$ como conjunto de variáveis, sendo V_{s_1} o símbolo inicial, e tem as regras de produção seguintes:

$$\begin{aligned}
V_{s_1} &\rightarrow \varepsilon \mid \mathtt{a}V_{s_2} \mid \mathtt{b}V_{s_2} \\
V_{s_2} &\rightarrow \mathtt{a}V_{s_3} \mid \mathtt{b}V_{s_3} \\
V_{s_3} &\rightarrow \varepsilon \mid \mathtt{a}V_{s_3} \mid \mathtt{b}V_{s_3}
\end{aligned}$$

As regras da gramática que se definiu na prova dada acima têm uma forma particular: no lado direito de cada regra ocorre no máximo uma variável; e se analisarmos as regras da gramática que têm alguma variável no lado direito,

vemos que a variável aparece sempre no fim (ou seja, o mais à direita possível) . As gramáticas desse tipo dizem-se lineares à direita.

Definição 15 *Uma gramática independente de contexto é* **linear à direita** *se todas as suas regras são da forma* $X \to w$ *ou* $X \to wY$*, em que* X *e* Y *são variáveis e* $w \in \Sigma^\star$*. E, é* **linear à esquerda** *se todas as suas regras são da forma* $X \to w$ *ou* $X \to Yw$*.*

Às gramáticas lineares à direita ou lineares à esquerda também se chama **gramáticas regulares**. Essa designação surge naturalmente como consequência dos resultados seguintes.

Proposição 36 *A classe de linguagens regulares é a classe de linguagens independentes de contexto geradas por gramáticas lineares à direita.*

Prova: A GIC que se obtém por conversão de um autómato finito pelo método que descrevemos acima é uma gramática linear à direita. Portanto, qualquer linguagem regular pode ser gerada por uma GIC linear à direita.

Resta mostrar que se L é uma LIC gerada por GIC $\mathcal{G} = (V, \Sigma, P, S)$ linear à direita, então L pode ser reconhecida por um autómato finito. Supomos, sem perda de generalidade, que as regras de $\mathcal{G}$ são da forma

$$\begin{array}{lcllcl} X & \to & aY & \qquad & X & \to & a \\ X & \to & \varepsilon & & X & \to & Y \end{array}$$

com $X, Y \in V$ quaisquer e $a \in \Sigma$. Se não, qualquer regra $X \to wY$, com $w = a_1 a_2 \ldots a_k$, para $a_i \in \Sigma$, podia ser substituída por k regras $X \to a_1 X_1$, $X_1 \to a_2 X_2$, $\ldots$, $X_{k-1} \to a_k Y$, por introdução de variáveis auxiliares novas.

O autómato finito que vamos definir, possivelmente um AFND-ε, é "parecido" com o que daria origem a $\mathcal{G}$. Começamos por transformar as regras de $\mathcal{G}$ de modo que apenas uma variável produza ε. Para isso, introduzimos uma nova variável X_f, e substituimos cada regra $X \to \varepsilon$ por $X \to X_f$ e cada regra $X \to a$ por $X \to aX_f$. Introduzimos ainda a regra $X_f \to \varepsilon$.

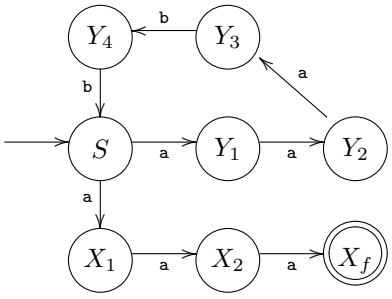
Definimos então um autómato finito $\mathcal{A} = (Q, \delta, e_S, F)$ assim. Para cada variável X da gramática, consideramos um estado e_X. Por cada regra $X \to aY$, com $a \in \Sigma$, consideramos uma transição (e_X, a, e_Y), e, por cada regra $X \to Y$, consideramos a transição (e_X, ε, e_Y). O conjunto de estados do autómato é $Q = \{\, e_X \mid X \in V \cup \{X_f\} \,\}$, o estado inicial é e_S (sendo S o símbolo inicial de $\mathcal{G}$), e o conjunto de estados finais $F = \{e_{X_f}\}$. Podemos verificar que este autómato reconhece $\mathcal{L}(\mathcal{G})$). □

Na **hierarquia de Chomsky**, as gramáticas regulares são designadas por gramáticas do Tipo 3, as independentes de contexto são Tipo 2, as dependentes de contexto são do Tipo 1, e as mais gerais são Tipo 0. Nas gramáticas dependentes de contexto, as regras são da forma $\alpha X \beta \to \alpha w \beta$, com $\alpha, \beta, w \in (V \cup \Sigma)^\star \setminus \{\varepsilon\}$, $w \neq \varepsilon$, pondendo ainda ter $S \to \varepsilon$, se S não ocorrer no lado direito de nenhuma regra.

Exemplo 109 Seja $\mathcal{G} = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, \{S \to \mathtt{aaa},\ S \to \mathtt{aaabb}S\}, S)$. Como $\mathcal{G}$ é linear à direita, podemos aplicar o método de conversão descrito na prova da Proposição 36 para obter um autómato finito que aceita $\mathcal{L}(\mathcal{G})$. Começamos por dar às produções a forma adequada, obtendo $\mathcal{G}' = (\{S, X_1, X_2, X_3, Y_1, Y_2, Y_3, Y_4, X_f\}, \{\mathtt{a}, \mathtt{b}\}, P, S)$, com P dado por:

$$
\begin{array}{llllllllllllll}
S & \to & \mathtt{a}X_1 & X_1 & \to & \mathtt{a}X_2 & X_2 & \to & \mathtt{a}X_f & & & & & \\
S & \to & \mathtt{a}Y_1 & Y_1 & \to & \mathtt{a}Y_2 & Y_2 & \to & \mathtt{a}Y_3 & Y_3 & \to & \mathtt{b}Y_4 & Y_4 & \to & \mathtt{b}S \\
X_f & \to & \varepsilon & & & & & & & & & & & &
\end{array}
$$

O autómato finito que se obtém é:



É interessante observar que se aplicassemos a este autómato o método de conversão de autómatos finitos para GICs lineares à direita, obteriamos $\mathcal{G}'$ (a menos das designações das variáveis).

Vamos ver agora que as GICs lineares à esquerda definem a mesma classe de linguagens que as GICs lineares à direita, e portanto, também elas caraterizam a classe das linguagens regulares. Por essa razão, as GICs lineares à direita e as GICs lineares à esquerda designam-se por gramáticas regulares.

Proposição 37 *A classe das linguagens que podem ser geradas por GICs lineares à direita é a classe de linguagens que pode ser geradas por GICs lineares à esquerda.*

Prova: Se uma dada linguagem L for gerada por uma gramática linear à esquerda, então se substituirmos cada regra dessa gramática da forma $X \to Yw$ por $X \to w^rY$ e cada regra da forma $X \to w$ por $X \to w^r$, com $w \in \Sigma^\star$, obtemos uma gramática linear à direita que gera L^r. Analogamente, se uma linguagem L for gerada por uma GIC linear à direita L então L^r é gerada por uma GIC linear à esquerda, pois bastaria substituir cada regra da forma $X \to wY$ por $X \to Yw^r$ e cada regra da forma $X \to w$ por $X \to w^r$, com $w \in \Sigma^\star$.

Logo, se L for gerada por uma gramática linear à esquerda, L^r é gerada por GIC linear à direita e, consequentemente, pela Proposição 36, concluimos que L^r é regular. Mas, se L^r é regular também L é regular e, novamente, pela Proposição 36, a linguagem L pode ser gerada por uma GIC linear à direita.

Por outro lado, se L for gerada por uma GIC linear à direita então L^r é regular e, consequentemente, pela Proposição 36, L^r pode ser gerada por uma GIC linear à direita, o que implica que $(L^r)^r$ pode ser gerada por uma GIC linear à esquerda. Como $(L^r)^r = L$, concluimos que se L for gerada por GIC linear à direita então L pode ser gerada por GIC linear à esquerda. □

Já vimos que existem linguagens independentes de contexto que não são regulares e que, portanto, não podem ser geradas por GICs lineares à direita (nem lineares à esquerda). Por exemplo, a linguagem constituída pelas sequências de parentesis curvos que são bem formadas. Também a linguagem $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$ de alfabeto $\{\mathtt{0},\mathtt{1}\}$ não é regular mas é independente de contexto, pois $L = \mathcal{L}(\mathcal{G})$ para $\mathcal{G} = (\{S\}, \{\mathtt{0},\mathtt{1}\}, \{S \to \varepsilon,\ S \to \mathtt{0}S\mathtt{1}\}, S)$.

Neste exemplo, bem como em vários anteriores, limitámo-nos a afirmar que $\mathcal{G}$ gerava a linguagem L dada, mas não o justificámos formalmente. Vamos agora ilustrar como se pode usar a relação de derivação para *tentar* provar que uma dada gramática gera uma certa linguagem. Escrevemos "tentar" porque não existe um algoritmo que possamos seguir efetuar a demonstração. Para L e $\mathcal{G}$ dadas, decidir se $\mathcal{L}(\mathcal{G}) = L$ é um problema indecidível (cf. Capítulo 7).

6.1.4 Exemplos de demonstração da correção de algumas gramáticas

Exemplo 110 A gramática $\mathcal{G} = (\{S\}, \{\mathtt{0},\mathtt{1}\}, \{S \to \varepsilon,\ S \to \mathtt{0}S\mathtt{1}\}, S)$ gera a linguagem $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$. Vamos provar formalmente esta afirmação.

Prova: Vamos mostrar que $L = \mathcal{L}(\mathcal{G})$, isto é, que, para todo $x \in \{\mathtt{0},\mathtt{1}\}^\star$, é equivalente dizer que $x \in \mathcal{L}(\mathcal{G})$ e dizer que $x = \mathtt{0}^n\mathtt{1}^n$, para algum $n \in \mathbb{N}$. Para isso, usando indução matemática, vamos provar que $S \Rightarrow_{\mathcal{G}}^n w$ se e só se $w = \mathtt{0}^n S \mathtt{1}^n$ ou $w = \mathtt{0}^{n-1} S \mathtt{1}^{n-1}$, quaisquer que seja $w \in \{S, \mathtt{0}, \mathtt{1}\}^\star$ e $n \geq 1$.

- (Caso de base) Para $n = 1$, temos $S \Rightarrow_{\mathcal{G}}^1 w$ se e só se $w = \varepsilon$ ou $w = \mathtt{0}S\mathtt{1}$, por definição de derivação imediata e por $\Rightarrow_{\mathcal{G}}^1$ ser $\Rightarrow_{\mathcal{G}}$, também por definição. Portanto, a condição verifica-se para $n = 1$.

- (Heriditariedade) Suponhamos que $S \Rightarrow_{\mathcal{G}}^k z$ se e só se $z = \mathtt{0}^{k-1}\mathtt{1}^{k-1}$ ou $z = \mathtt{0}^k S \mathtt{1}^k$, para $k \geq 1$ fixo e z qualquer. Por definição de $\Rightarrow_{\mathcal{G}}^{k+1}$, tem-se $S \Rightarrow_{\mathcal{G}}^{k+1} w$ se e só se $S \Rightarrow_{\mathcal{G}}^k z \ \wedge\ z \Rightarrow_{\mathcal{G}}^1 w$, para algum $z \in \{S, \mathtt{0}, \mathtt{1}\}^\star$ tal que S ocorre em z, qualquer que seja $w \in \{S, \mathtt{0}, \mathtt{1}\}^\star$. Mas, pela hipótese de indução, $S \Rightarrow_{\mathcal{G}}^k z$ e S ocorre em z se e só se $z = \mathtt{0}^k S \mathtt{1}^k$. E, consequentemente, de $z \Rightarrow_{\mathcal{G}}^1 w$ sse $w = \mathtt{0}^k \varepsilon \mathtt{1}^k$ ou $w = \mathtt{0}^k \mathtt{0} S \mathtt{1}\mathtt{1}^k$, por aplicação das regras da gramática (respetivamente, $S \to \varepsilon$ e $S \to \mathtt{0}S\mathtt{1}$). Portanto, como $\mathtt{0}^k\varepsilon\mathtt{1}^k = \mathtt{0}^k\mathtt{1}^k$ e $\mathtt{0}^k\mathtt{0}S\mathtt{1}\mathtt{1}^k = \mathtt{0}^{k+1}S\mathtt{1}^{k+1}$, concluimos que a condição é hereditária. Ou seja, sob a hipótese, tem-se $S \Rightarrow_{\mathcal{G}}^{k+1} w$ se e só se $w = \mathtt{0}^k\mathtt{1}^k$ ou e $w = \mathtt{0}^{k+1}S\mathtt{1}^{k+1}$.

Por indução matemática, concluimos que $S \Rightarrow_{\mathcal{G}}^{n+1} w$ se e só se $w = \mathtt{0}^n\mathtt{1}^n$ ou $w = \mathtt{0}^{n+1}S\mathtt{1}^{n+1}$, para todo $n \in \mathbb{N}$ e todo $w \in \{S, \mathtt{0}, \mathtt{1}\}^\star$. Como $\Rightarrow_{\mathcal{G}}^\star$ é dada por $\bigcup_{n\in\mathbb{N}}(\Rightarrow_{\mathcal{G}}^n)$, segue $\mathcal{L}(\mathcal{G}) = L$, pois, em particular, concluimos que, quando x é uma sequência de terminais, $S \Rightarrow_{\mathcal{G}}^{n+1} x$ se e só se $x = \mathtt{0}^n\mathtt{1}^n$, com $n \in \mathbb{N}$. □

Exemplo 111 Seja $\mathcal{G} = (\{S, X, R\}, \{\mathtt{0}, \mathtt{1}\}, P, S)$ a gramática definida no Exemplo 98, com P dado por:

$$S \;\to\; \mathtt{0}\,X\,R \qquad\qquad R \;\to\; \mathtt{0}\,X\,R \mid \mathtt{0} \qquad\qquad X \;\to\; \mathtt{11}\,X \mid \mathtt{11}$$

Vamos demonstrar que $\mathcal{L}(\mathcal{G}) = \{\mathtt{0}x_1 \ldots \mathtt{0}x_k\mathtt{0} \mid k \geq 1, x_i \in \mathcal{L}(\ (\mathtt{11})^\star\mathtt{11}\ ) \text{ para } 1 \leq i \leq k\}$.

- A linguagem gerada a partir de X é $\mathcal{L}(\ (\mathtt{11})^\star\mathtt{11}\ ) = \{\mathtt{1}^{2n} \mid n \geq 1\}$.

  Prova: Vamos mostrar que $X \Rightarrow_{\mathcal{G}}^{n} w$ se e só se $w = \mathtt{1}^{2n}$ ou $w = \mathtt{1}^{2n}X$, para todo $n \geq 1$ e $w \in (\Sigma \cup V)^\star$.

  - (caso de base, $n = 1$) É trivial que se a condição se verifica para $n = 1$, pois, por definição de $\Rightarrow_{\mathcal{G}}^{1}$ e das X-produções, temos $X \Rightarrow_{\mathcal{G}}^{1} w$ se e só se $w = \mathtt{11}$ ou $w = \mathtt{11}X$.
  - (hereditariedade) Como hipótese de indução, suponhamos que, para $n \geq 1$ fixo, é verdade que $X \Rightarrow_{\mathcal{G}}^{n} w$ se e só se $w = \mathtt{1}^{2n}$ ou $w = \mathtt{1}^{2n}X$, para todo $w \in (\Sigma \cup V)^\star$. Por definição de $\Rightarrow_{\mathcal{G}}^{n+1}$, para todo $x \in (\Sigma \cup V)^\star$, tem-se $X \Rightarrow_{\mathcal{G}}^{n+1} x$ se e só se existe $x_1 \in (\Sigma \cup V)^\star$ tal que x_1 tem alguma variável, $X \Rightarrow_{\mathcal{G}}^{n} x_1$ e $x_1 \Rightarrow_{\mathcal{G}}^{1} x$. Então, pela hipótese de indução, concluimos que $x_1 = \mathtt{1}^{2n}X$ e, como $x_1 \Rightarrow_{\mathcal{G}}^{1} x$, obtemos $x = \mathtt{1}^{2n}\mathtt{11}$ ou $x = \mathtt{1}^{2n}\mathtt{11}X$, pela definição de derivação imediata. Logo, sob a hipótese de indução, $X \Rightarrow_{\mathcal{G}}^{n+1} x$ se e só se $x = \mathtt{1}^{2(n+1)}$ ou $x = \mathtt{1}^{2(n+1)}X$.
  - Portanto, pelo princípio de indução, concluimos que $X \Rightarrow_{\mathcal{G}}^{n} w$ se e só se $w = \mathtt{1}^{2n}$ ou $w = \mathtt{1}^{2n}X$, para todo $w \in (\Sigma \cup V)^\star$. E, em particular, para $w \in \Sigma^\star$, tem-se $X \Rightarrow_{\mathcal{G}}^{n} w$ se e só se $w = \mathtt{1}^{2n}$, ou seja, $X \Rightarrow_{\mathcal{G}}^{n} w$ se e só se $w \in \{\mathtt{1}^{2n} \mid n \geq 1\}$.

- A linguagem gerada a partir de R é $\{\mathtt{0}\mathtt{1}^{2n} \mid n \geq 1\}^\star\{\mathtt{0}\} = \mathcal{L}(\ (\mathtt{0}(\mathtt{11})^\star\mathtt{11})^\star\mathtt{0}\ )$

  Prova: Como as X-produções não geram nem R nem S, vamos analisar derivações a partir de R em que não são aplicadas X-produções e mostrar que $R \Rightarrow_{\mathcal{G}}^{n} w$ *sem usar X-produções* se e só se $w = (\mathtt{0}X)^n R$ ou $w = (\mathtt{0}X)^{n-1}\mathtt{0}$, para todo $n \geq 1$ e $w \in (\Sigma \cup V)^\star$.

  - (caso de base, $n = 1$) Por definição de $\Rightarrow_{\mathcal{G}}^{1}$, temos $R \Rightarrow_{\mathcal{G}}^{1} w$ se e só se $w = \mathtt{0}XR$ ou $w = \mathtt{0}$. Portanto, como $\mathtt{0}XR = (\mathtt{0}X)^1 R$ e $\mathtt{0} = (\mathtt{0}X)^0\mathtt{0}$, é trivial que acondição se verifica para $n = 1$.
  - (hereditariedade) Como hipótese de indução, suponhamos que, para $n \geq 1$ fixo, para derivações que não usam X-produções, é verdade que $R \Rightarrow_{\mathcal{G}}^{n} w$ se e só se $w = (\mathtt{0}X)^n R$ ou $w = (\mathtt{0}X)^{n-1}\mathtt{0}$, para todo $w \in (\Sigma \cup V)^\star$. Como $R \Rightarrow_{\mathcal{G}}^{n+1} x$ se e só se existe $x_1 \in (\Sigma \cup V)^\star$ tal que x_1 tem alguma variável, $R \Rightarrow_{\mathcal{G}}^{n} x_1$ e $x_1 \Rightarrow_{\mathcal{G}}^{1} x$, então, não sendo usadas X-produções, concluimos que $x_1 = (\mathtt{0}X)^n R$ ou $x_1 = (\mathtt{0}X)^{n-1}\mathtt{0}$, pela hipótese de indução. Então, como X não pode ser substituída na derivação $x_1 \Rightarrow_{\mathcal{G}}^{1} x$, concluimos que $x_1 = (\mathtt{0}X)^n R$ e, consequentemente, $x = (\mathtt{0}X)^{n+1}R$ ou $w = (\mathtt{0}X)^n\mathtt{0}$.
  - Portanto, pelo princípio de indução, concluimos que $R \Rightarrow_{\mathcal{G}}^{n} w$ sem usar X-produções $x = (\mathtt{0}X)^n R$ ou $w = (\mathtt{0}X)^{n-1}\mathtt{0}$, para todo $n \geq 1$.

Para concluir a prova de que a linguagem gerada a partir de R é $\{\mathtt{0}\mathtt{1}^{2n} \mid n \geq 1\}^\star\{\mathtt{0}\} = \mathcal{L}(\ (\mathtt{0}(\mathtt{11})^\star\mathtt{11})^\star\mathtt{0}\ )$, comecemos por observar que, se $w \in \Sigma^\star$ e $R \Rightarrow_{\mathcal{G}}^{n+k} w$, sendo n o número de R-produções aplicadas e k o número das restantes, com $n \geq 1$ e $k \geq n-1$, então, como X não gera R nem S, concluimos que $R \Rightarrow_{\mathcal{G}}^{n+k} w$ sse $R \Rightarrow_{\mathcal{G}}^{n} (\mathtt{0}X)^{n-1}\mathtt{0} \Rightarrow_{\mathcal{G}}^{k} w$, onde os k últimos passos resultam de aplicações de X-produções. Como

$$(\mathtt{0}X)^{n-1} = \underbrace{\mathtt{0}X\mathtt{0}X\cdots\mathtt{0}X}_{n-1 \text{ vezes}}$$

então $w = \mathtt{0}w_1\mathtt{0}w_2\cdots\mathtt{0}w_{n-1}\mathtt{0}$, para w_i pertencente à linguagem gerada a partir de X, para $0 \leq i \leq n-1$. Considerando a prova anterior, sabemos que tal linguagem é $\{\mathtt{1}^{2p} \mid p \geq 1\}$ e que $X \Rightarrow_{\mathcal{G}}^{p} \mathtt{1}^{2p}$. Portanto, $w = \mathtt{0}\mathtt{1}^{2k_1}\mathtt{0}\mathtt{1}^{2k_2}\cdots\mathtt{0}\mathtt{1}^{2k_{n-1}}\mathtt{0}$, com $k_1 + k_2 + \cdots + k_{n-1} = k$, sendo $k \geq n-1 \geq 0$.

- Para concluir que $\mathcal{L}(\mathcal{G}) = \{\mathtt{0}x_1\ldots\mathtt{0}x_k\mathtt{0} \mid k \geq 1, x_i \in \mathcal{L}(\ (\mathtt{11})^\star\mathtt{11}\ ) \text{ para } 1 \leq i \leq k\}$, resta analisar as derivações a partir de S. Como $S \Rightarrow_{\mathcal{G}}^\star w$ se e só se $S \Rightarrow_{\mathcal{G}} \mathtt{0}XR \Rightarrow_{\mathcal{G}}^\star \mathtt{0}xR \Rightarrow_{\mathcal{G}}^\star \mathtt{0}xy = w$, com $X \Rightarrow_{\mathcal{G}}^\star x$ e $R \Rightarrow_{\mathcal{G}}^\star y$, segue das provas anteriores que $\mathcal{L}(\mathcal{G})$ é a linguagem indicada. □

Exemplo 112 A linguagem $L = \{\mathtt{aaa}\,x\,\mathtt{a}\,y \mid |x| \leq 2|y|,\ \ x,y \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star\}$ de alfabeto $\{\mathtt{0},\mathtt{1},\mathtt{b},\mathtt{a}\}$ não é regular mas é independente de contexto. Vamos provar formalmente esta afirmação.

- L **não é regular**, pois não satisfaz a condição do Lema da repetição (para linguagens regulares).

  De facto, seja $n \in \mathbb{N} \setminus \{0\}$. Procuremos $x \in L$, com $|x| \geq n$, e tal que, para toda a decomposição de x como $x = uvw$, com $|uv| \leq n \wedge v \neq \varepsilon$, existe $i \in \mathbb{N}$ tal que $uv^iw \notin L$. Tomemos, por exemplo, $x = \mathtt{aaa}\mathtt{1}^{2n}\mathtt{a}\mathtt{1}^n$, pois sabemos que o número de 1's em qualquer prefixo uv de x, com $|uv| \leq n$, não poderia ser aumentado. Podemos afirmar que, se $|uv| \leq n \wedge v \neq \varepsilon$, então uv é prefixo de $\mathtt{aaa}\mathtt{1}^{n-3}$, se $n \geq 3$, ou uv é prefixo de $\mathtt{aaa}$, se $n < 3$. Como $v \neq \varepsilon$, então v tem algum a ou v não tem a's (mas terá 1's, considerando a escolha de x).

  Se v tiver algum a, cortamos v (tomando $i = 0$) e $uv^0w \notin L$ porque não começa por $\mathtt{aaa}$.

  Se v não tiver a's então só tem 1's e, como uv é prefixo de $\mathtt{aaa}\mathtt{1}^{n-3}$, temos

  $$x = \underbrace{\mathtt{aaa}\mathtt{1}^k}_{u}\ \underbrace{\mathtt{1}^{|v|}}_{v}\ \underbrace{\mathtt{1}^{2n-k-|v|}\mathtt{a}\mathtt{1}^n}_{w}$$

  e vemos que, por exemplo, para $i = 2$, a palavra $uv^iw = uvvw = \mathtt{aaa}\mathtt{1}^{2n+|v|}\mathtt{a}\mathtt{1}^n \notin L$. Portanto, L não é regular.

- A linguagem $L = \{\mathtt{aaa}\,x\,\mathtt{a}\,y \mid |x| \leq 2|y|,\ \ x,y \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star\}$ **é independente de contexto** porque é gerada, por exemplo, pela GIC $\mathcal{G} = (\{S,R,A\},\{\mathtt{0},\mathtt{1},\mathtt{a},\mathtt{b}\},P,S)$, cujas regras são:

  $$\begin{array}{lcl} S & \rightarrow & \mathtt{aaa}\,R \\ R & \rightarrow & A\,A\,R\,A \mid A\,R\,A \mid R\,A \mid \mathtt{a} \\ A & \rightarrow & \mathtt{b} \mid \mathtt{1} \mid \mathtt{0} \end{array}$$

Vamos demonstrar formalmente que $\mathcal{G}$ gera L, ou seja $\mathcal{L}(\mathcal{G}) = L$.

– Provemos que $\mathcal{L}(\mathcal{G}) \subseteq L$, isto é, que para todo $x \in \Sigma^\star$, se $S \Rightarrow_{\mathcal{G}}^\star x$ então $x \in L$.

Como a linguagem gerada a partir de A é $\{\mathtt{0}, \mathtt{1}, \mathtt{b}\}$, podemos transformar qualquer derivação numa derivação em que as A-produções são usadas apenas nos últimos passos. Assim, começamos por analisar as formas que se podem derivar de S sem substituir A. De facto, focaremos a análise nas formas que se podem derivar de R uma vez que, qualquer derivação a partir de S, começa por $S \Rightarrow_{\mathcal{G}} \mathtt{aaa}R$. Vamos provar que, para todo $n \geq 1$ e $x \in (V \cup \Sigma)^\star$, se $R \Rightarrow_{\mathcal{G}}^n x$ sem usar A-produções na derivação, então $x = A^k\, \mathtt{a}\, A^{n-1}$ com $0 \leq k \leq 2(n-1)$ **ou** $x = A^k\, R\, A^n$ com $0 \leq k \leq 2n$. Tal permitirá concluir que, as palavras $x \in \Sigma^\star$ geradas a partir de R são da forma $\alpha\mathtt{a}\beta$, com $|\alpha| \leq 2|\beta|$, $\alpha, \beta \in \{\mathtt{0}, \mathtt{1}, \mathtt{b}\}^\star$.

A prova será por indução sobre o número de passos da derivação.

∗ (Caso de base, $n = 1$). Se $R \Rightarrow^1 x$ então $x = AARA$ ou $x = ARA$ ou $x = RA$ ou $x = \mathtt{a}$. Logo, é verdade que se $R \Rightarrow^1 x$ então $x = A^0\, \mathtt{a}\, A^0$ ou $x = A^k\, R\, A$, com $0 \leq k \leq 2$.

∗ (Heriditariedade) Por definição de $\Rightarrow_{\mathcal{G}}^{n+1}$, tem-se $R \Rightarrow_{\mathcal{G}}^{n+1} x$ se e só se $R \Rightarrow_{\mathcal{G}}^n x' \wedge x' \Rightarrow_{\mathcal{G}} x$, para algum $x' \in (V \cup \Sigma)^\star$. Portanto, se considerarmos derivações sem substituição de A, na derivação de x e de x' não serão usadas A-produções.

Assim, como hipótese de indução, suponhamos que, para todo $w \in (V \cup \Sigma)^\star$, se $R \Rightarrow^n w$ sem usar A-produções, então $w = A^k R A^n$, com $0 \leq k \leq 2n$, **ou** $w = A^k \mathtt{a} A^{n-1}$, com $0 \leq k \leq 2(n-1)$. Então, se tivermos $R \Rightarrow_{\mathcal{G}}^n x' \Rightarrow_{\mathcal{G}}^n x$, podemos afirmar que $x' = A^k R A^n$, para algum k tal que $0 \leq k \leq 2n$, pela hipótese e por não podermos usar A-produções no último passo desta derivação.

Como x resulta de $x' = A^k R A^n$ por aplicação de uma R-produção, então x terá a forma esperada, ou seja,

$$x = A^q\, \mathtt{a}\, A^n, \text{ para algum } q, \text{ com } 0 \leq q \leq 2n$$

ou

$$x = A^q\, R\, A^{n+1}, \text{ para algum } q, \text{ com } 0 \leq q \leq 2n+2.$$

De facto, considerando as quatro regras possíveis para R, de $x' \Rightarrow_{\mathcal{G}} x$, concluimos que

$x = A^{k+2} R A^{n+1}$ por $x' = A^k R A^n \Rightarrow_{\mathcal{G}} A^k AARA^{n+1}$, ou

$x = A^{k+1} R A^{n+1}$ por $x' = A^k R A^n \Rightarrow_{\mathcal{G}} A^k ARA^{n+1}$, ou

$x = A^k R A^{n+1}$ por $x' = A^k R A^n \Rightarrow_{\mathcal{G}} A^k R A^{n+1}$, ou

$x = A^k \mathtt{a} A^n$ por $x' = A^k R A^n \Rightarrow_{\mathcal{G}} A^k \mathtt{a} A^n$,

e, sendo $0 \leq k \leq 2n$, tem-se

$$x = A^q\, R\, A^{n+1}, \text{ para } 0 \leq q \leq k+2 \leq 2(n+1)$$

ou

$$x = A^q\, \mathtt{a}\, A^n, \text{ para } 0 \leq q \leq 2n$$

  - as palavras $w \in \Sigma^\star$ geradas a partir de R são da forma $\alpha\mathtt{a}\beta$, com $|\alpha| \leq 2|\beta|$, $\alpha, \beta \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star$, porque, quando aplicarmos as A-produções para substituir os A's, teremos $R \Rightarrow_{\mathcal{G}}^{n} A^k\mathtt{a}A^{n-1} \Rightarrow_{\mathcal{G}}^{k+n-1} \alpha\mathtt{a}\beta = w$, para algum $n \geq 1$ e k tal que $0 \leq k \leq 2(n-1)$, sendo $|\alpha| = k$ e $|\beta| = n-1$. Portanto, se $S \Rightarrow_{\mathcal{G}}^{\star} x$ então x é da forma $\mathtt{aaa}x_1\mathtt{a}x_2$, com $|x_1| \leq 2|x_2|$ e $x_1, x_2 \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star$. Logo, $x \in \mathcal{L}(\mathcal{G})$.

- Reciprocamente, vamos provar que $\mathcal{L}(\mathcal{G}) \supseteq L$, isto é, $S \Rightarrow_{\mathcal{G}}^{\star} \mathtt{aaa}x\mathtt{a}y$, para quaisquer $x, y \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star$, tais que $|x| \leq 2|y|$. Como $S \Rightarrow_{\mathcal{G}} \mathtt{aaa}R$, basta ver que qualquer palavra da forma $x\mathtt{a}y$, com x e y nessas condições, pode ser derivada a partir de R. Denotando $|x|$ e $|y|$ por k e n, vemos que:

  - se $2n \geq k > n$, aplicamos $k-n$ vezes $R \to A\,A\,R\,A$ e, em seguida, $2n-k$ vezes $R \to A\,R\,A$, para obter $A^{|x|}\,R\,A^{|y|}$. Finalmente, aplicamos uma vez a regra $R \to \mathtt{a}$ e, para concluir, $k+n$ vezes A-produções para obter x e y. Ou seja,

$$R \Rightarrow_{\mathcal{G}}^{k-n} \quad A^{2(k-n)}\,R\,A^{k-n} \quad \Rightarrow_{\mathcal{G}}^{2n-k} \quad A^k\,R\,A^n \quad \Rightarrow_{\mathcal{G}} \quad A^k\,\mathtt{a}\,A^n \quad \Rightarrow_{\mathcal{G}}^{n+k} x\mathtt{a}y$$

  - se $k < n$, aplicamos k vezes $R \to A\,R\,A$ e $n-k$ vezes $R \to R\,A$, e depois $R \to \mathtt{a}$, e finalmente $n+k$ vezes A-produções (para derivar x e y). ou seja,

$$R \Rightarrow_{\mathcal{G}}^{k} \quad A^k\,R\,A^k \quad \Rightarrow_{\mathcal{G}}^{n-k} \quad A^k\,R\,A^n \quad \Rightarrow_{\mathcal{G}} \quad A^k\,\mathtt{a}\,A^n \quad \Rightarrow_{\mathcal{G}}^{n+k} x\mathtt{a}y$$

  - se $k = n$, aplicamos n vezes a regra $R \to A\,R\,A$, para obter $A^{|x|}\,R\,A^{|y|}$, depois uma vez a regra $R \to \mathtt{a}$ e, para concluir, $2n$ vezes regras para A de modo a gerar x e y. Ou seja,

$$R \Rightarrow_{\mathcal{G}}^{n} \quad A^n\,R\,A^n \quad \Rightarrow_{\mathcal{G}} \quad A^n\,\mathtt{a}\,A^n \quad \Rightarrow_{\mathcal{G}}^{2n} x\mathtt{a}y$$

- Como mostrámos que $\mathcal{L}(\mathcal{G}) \subseteq L$ e $\mathcal{L}(\mathcal{G}) \supseteq L$, segue $\mathcal{L}(\mathcal{G}) = L$. Logo, L é independente de contexto. □

6.2 Conversão entre GICs e autómatos de pilha

Nesta secção mostramos que a classe das linguagens independentes de contexto é a classe das linguagens que são reconhecidas por autómatos de pilha. Na prova de que qualquer linguagem independente de contexto pode ser reconhecida por um autómato de pilha, vamos partir de uma gramática auxiliar, dada na forma normal de Greibach.

Definição 16 *Uma gramática independente de contexto $\mathcal{G} = (V, \Sigma, P, S)$ diz-se na* **forma normal de Greibach** *sse qualquer produção é da forma $A \to aw$, com $A \in V$, $a \in \Sigma$ e $w \in V^\star$, quaisquer.*

O exemplo seguinte mostra que, em alguns casos, é simples encontrar uma gramática na forma normal de Greibach equivalente a uma gramática dada. Contudo, por vezes, a conversão para a forma de Greibach não é tão imediata. Na

secção 6.3.6, provamos a Proposição 41 que descreve um método genérico para obter uma gramática na forma de Greibach a partir de uma gramática dada (para uma linguagem L tal que $\varepsilon \notin L$).

Exemplo 113 A gramática $\mathcal{G} = (\{S\}, \{\texttt{f}, \texttt{x}, \texttt{(}, \texttt{)}, \texttt{,}\}, P, S)$, com produções

$$\begin{aligned} S &\to \texttt{x} \\ S &\to \texttt{f(}S\texttt{,}S\texttt{)} \end{aligned}$$

pode ser facilmente convertida à forma normal de Greibach. Basta introduzir novas variáveis para representar terminais e modificar as produções assim:

$$\begin{aligned} S &\to \texttt{x} \\ S &\to \texttt{f}ASBSC \\ A &\to \texttt{(} \\ B &\to \texttt{,} \\ C &\to \texttt{)} \end{aligned}$$

Proposição 38 *Qualquer linguagem independente de contexto pode ser reconhecida por algum autómato de pilha.*

Prova: Seja L uma linguagem independente de contexto de alfabeto Σ. Pela proposição 41 (que provaremos na secção 6.3.6), existe uma gramática $\mathcal{G} = (V, \Sigma, P, S)$ na forma formal de Greibach que gera $L \setminus \{\varepsilon\}$. Por conversão de $\mathcal{G}$, podemos obter um autómato de pilha que aceita $L \setminus \{\varepsilon\}$ por pilha vazia. Esse autómato é $\mathcal{A} = (\{q\}, \Sigma, V, \delta, q, S, \{\ \})$, com $\delta(q, a, A) = \{(q, \beta) \mid (A \to a\beta) \in P\}$, para $a \in \Sigma$ e $A \in V$.

De facto, por indução sobre n, podemos mostrar que $S \Rightarrow^n_{E,\mathcal{G}} x\beta$ se e só se $(q, x, S) \vdash^n_{\mathcal{A}} (q, \varepsilon, \beta)$, para todo $n \in \mathbb{N}$, $\beta \in V^\star$ e $x \in \Sigma^\star$, onde $\Rightarrow_{E,\mathcal{G}}$ denota derivação pela esquerda.

Logo, $S \Rightarrow^\star_{\mathcal{G}} x$ se e só se $(q, x, S) \vdash^\star_{\mathcal{A}} (q, \varepsilon, \varepsilon)$, para todo $x \in \Sigma^\star$.

Se $\varepsilon \in L$, então o autómato de pilha $\mathcal{A}' = (\{q\}, \Sigma, V, \delta', q, S, \{\ \})$, com $\delta'(q, a, A) = \delta(q, a, A)$ para todo $a \in \Sigma$ e todo $A \in V$, e $\delta'(q, \varepsilon, S) = \{(q, \varepsilon)\}$ reconhece L por pilha vazia. □

Exemplo 114 Na continuação do Exemplo 113, partindo da gramática na forma normal de Greibach, aplicando o método descrito na prova da Proposição 38, obtemos um autómato de pilha $\mathcal{A} = (q, \{\texttt{f}, \texttt{x}, \texttt{(}, \texttt{)}, \texttt{,}\}, \{\texttt{S}, \texttt{A}, \texttt{B}, \texttt{C}\}, \delta, q, \emptyset)$, que aceita $\mathcal{L}(\mathcal{G})$ por pilha vazia. Mostramos abaixo a correspondência entre as regras de $\mathcal{G}'$ e a função de transição δ.

$$\begin{aligned} S &\to \texttt{x} && \delta(q, \texttt{x}, \texttt{S}) = \{(q, \varepsilon)\} \\ S &\to \texttt{f}ASBSC && \delta(q, \texttt{f}, \texttt{S}) = \{(q, \texttt{ASBSC})\} \\ A &\to \texttt{(} && \delta(q, \texttt{(}, \texttt{A}) = \{(q, \varepsilon)\} \\ B &\to \texttt{,} && \delta(q, \texttt{,}, \texttt{B}) = \{(q, \varepsilon)\} \\ C &\to \texttt{)} && \delta(q, \texttt{)}, \texttt{C}) = \{(q, \varepsilon)\} \end{aligned}$$

Exemplo 115 Seja $\mathcal{G}$ a gramática para a linguagem das expressões regulares sobre $\{\mathtt{a}, \mathtt{b}\}$, analisada no Exemplo 101:

$$\begin{array}{rcl} S & \rightarrow & \mathtt{a} \mid \mathtt{b} \mid \emptyset \mid \boxed{\varepsilon} \\ S & \rightarrow & (S{+}S) \mid (SS) \mid (S^{\star}) \end{array}$$

Também neste caso é facil converter $\mathcal{G}$ à forma normal de Greibach, e daí obter um autómato de pilha que reconhece a linguagem das expressões regulares, por pilha vazia:

$S \rightarrow \mathtt{a} \mid \mathtt{b} \mid \emptyset \mid \boxed{\varepsilon}$		$\delta(q, \mathtt{a}, \mathtt{S}) = \{(q, \varepsilon)\} = \delta(q, \mathtt{b}, \mathtt{S}) = \delta(q, \emptyset, \mathtt{S}) = \delta(q, \boxed{\varepsilon}, \mathtt{S})$
$S \rightarrow (SOSB \mid (SSB \mid (SEB$		$\delta(q, \mathtt{(}, \mathtt{S}) = \{(q, \mathtt{SOSB}), (q, \mathtt{SSB}), (q, \mathtt{SEB})\}$
$O \rightarrow +$		$\delta(q, +, \mathtt{O}) = \{(q, \varepsilon)\}$
$B \rightarrow)$		$\delta(q, \mathtt{)}, \mathtt{B}) = \{(q, \varepsilon)\}$
$E \rightarrow \star$		$\delta(q, \star, \mathtt{E}) = \{(q, \varepsilon)\}$

Para ilustrar a correspondência entre derivações na gramática normalizada e alterações de configuração no autómato, analisemos, por exemplo, o processamento de $x = \mathtt{(((aa)+b)^{\star})b)}$:

$$\begin{array}{rcl@{\qquad}rcl} S & \Rightarrow & (SSB & (q, x, \mathtt{S}) & \vdash & (q, \underline{\mathtt{(((aa)+b)^{\star})b)}}, \mathtt{SSB}) \\ & \Rightarrow & ((SEBSB & & \vdash & (q, \underline{\mathtt{((aa)+b)^{\star})b)}}, \mathtt{SEBSB}) \\ & \Rightarrow & (((SOSBEBSB & & \vdash & (q, \underline{\mathtt{(aa)+b)^{\star})b)}}, \mathtt{SOSBEBSB}) \\ & \Rightarrow & ((((SSBOSBEBSB & & \vdash & (q, \underline{\mathtt{aa)+b)^{\star})b)}}, \mathtt{SSBOSBEBSB}) \\ & \Rightarrow & ((((\mathtt{a}SBOSBEBSB & & \vdash & (q, \underline{\mathtt{a)+b)^{\star})b)}}, \mathtt{SBOSBEBSB}) \\ & \Rightarrow & ((((\mathtt{aa}BOSBEBSB & & \vdash & (q, \underline{\mathtt{)+b)^{\star})b)}}, \mathtt{BOSBEBSB}) \\ & \Rightarrow & ((((\mathtt{aa})OSBEBSB & & \vdash & (q, \underline{\mathtt{+b)^{\star})b)}}, \mathtt{OSBEBSB}) \\ & \Rightarrow & ((((\mathtt{aa})+SBEBSB & & \vdash & (q, \underline{\mathtt{b)^{\star})b)}}, \mathtt{SBEBSB}) \\ & \Rightarrow & ((((\mathtt{aa})+\mathtt{b}BEBSB & & \vdash & (q, \underline{\mathtt{)^{\star})b)}}, \mathtt{BEBSB}) \\ & \Rightarrow & ((((\mathtt{aa})+\mathtt{b})EBSB & & \vdash & (q, \underline{\mathtt{^{\star})b)}}, \mathtt{EBSB}) \\ & \Rightarrow & ((((\mathtt{aa})+\mathtt{b})^{\star}BSB & & \vdash & (q, \underline{\mathtt{)b)}}, \mathtt{BSB}) \\ & \Rightarrow & ((((\mathtt{aa})+\mathtt{b})^{\star})SB & & \vdash & (q, \underline{\mathtt{b)}}, \mathtt{SB}) \\ & \Rightarrow & ((((\mathtt{aa})+\mathtt{b})^{\star})\mathtt{b}B & & \vdash & (q, \underline{\mathtt{)}}, \mathtt{B}) \\ & \Rightarrow & ((((\mathtt{aa})+\mathtt{b})^{\star})\mathtt{b}) = x & & \vdash & (q, \varepsilon, \varepsilon) \end{array}$$

Proposição 39 *Qualquer linguagem que possa ser reconhecida por um autómato de pilha é independente de contexto.*

Prova: Suponhamos que $\mathcal{A}$ é um autómato de pilha que reconhece a linguagem L. Vimos que a classe de linguagens aceites por autómatos de pilha por estados finais é a classe de linguagens aceites por autómatos de pilha por

pilha vazia. Assim, sem perda de generalidade, podemos considerar que L é aceite por $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{\ \})$ por pilha vazia. O resto da prova consiste na descrição de um algoritmo que, dado um autómato de pilha $\mathcal{A}$, com aceitação por *pilha vazia*, determina uma gramática independente que gera $\mathcal{L}(\mathcal{A})$.

Tem-se $x \in \mathcal{L}(\mathcal{A})$ se e só se $(q_0, x, Z_0) \vdash_{\mathcal{A}}^{\star} (q_0, \varepsilon, \varepsilon)$, ou seja, se partindo do estado q_0, e tendo Z_0 na pilha, o autómato conseguir chegar a pilha vazia, tendo processado completamente x.

Vamos definir uma gramática independente de contexto cujas variáveis, com excepção do símbolo inicial, são representadas por ternos $[q, Z, q']$, em que $q, q' \in Q$ e $Z \in \Gamma$, cuja ideia é a seguinte. Se o autómato $\mathcal{A}$ estiver no estado q e tiver Z no topo da pilha, então existe um conjunto de palavras (possivelmente $\{\ \}$) que o levam ao estado q' e retiram Z da pilha. O símbolo inicial da gramática vai ser S e as regras da gramática são as seguintes:

- para todo $q \in Q$, inclui-se a regra $S \to [q_0, Z_0, q]$, significando que as palavras da linguagem são aquelas que levam o autómato do estado inicial a algum estado, retirando Z_0 da pilha;
- se $(q', \varepsilon) \in \delta(q, a, Z)$, inclui-se a regra $[q, Z, q'] \to a$, quaisquer que sejam $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$ e $q, q' \in Q$;
- se $(q', X_1 \cdots X_n) \in \delta(q, a, Z)$, inclui-se regras $[q, Z, q_n] \to a[q', X_1, q_1] \cdots [q_{n-1}, X_n, q_n]$, para todos os estados $q_1, \ldots, q_{n-1}, q_n \in Q$, quaisquer que sejam $X_1, \ldots X_n \in \Gamma$, $a \in \Sigma \cup \{\varepsilon\}$ e $q, q' \in Q$.

Explorando a relação entre $\vdash_{\mathcal{A}}$ e $\Rightarrow_{\mathcal{G}}$, não é difícil concluir que a gramática assim definida gera a mesma linguagem que o autómato dado reconhece. Omitiremos os detalhes ténicos da demonstração desse facto. □

O Exemplo 116 ilustra a aplicação do método descrito na prova da Proposição 39 para conversão de um autómato de pilha, com aceitação por pilha vazia, numa GIC equivalente (i.e., que gera a linguagem que o autómato aceita).

Exemplo 116 Seja $\mathcal{A} = (\{s_0, s_1\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{Z}, \mathtt{A}\}, \delta, s_0, \mathtt{Z}, \{\ \})$ um autómato de pilha, com aceitação por pilha vazia, onde δ é dada por:

$$\begin{array}{lcllcl} \delta(s_0, \varepsilon, \mathtt{Z}) & = & \{(s_1, \mathtt{Z})\} & \delta(s_0, \mathtt{0}, \mathtt{Z}) & = & \{(s_0, \mathtt{AZ})\} \\ \delta(s_0, \mathtt{0}, \mathtt{A}) & = & \{(s_0, \mathtt{AA})\} & \delta(s_0, \mathtt{1}, \mathtt{A}) & = & \{(s_1, \varepsilon)\} \\ \delta(s_1, \mathtt{1}, \mathtt{A}) & = & \{(s_1, \varepsilon)\} & \delta(s_1, \mathtt{1}, \mathtt{Z}) & = & \{(s_1, \varepsilon), (s_1, \mathtt{Z})\} \end{array}$$

As variáveis da gramática equivalente, definida na prova da Proposição 39, são todos os ternos $[s, X, s']$, com $s, s' \in \{s_0, s_1\}$ e $X \in \{\mathtt{Z}, \mathtt{A}\}$, e ainda S (o símbolo inicial). A variável $[s, X, s']$ representa todas as palavras x tais que $(s, x, X) \vdash_{\mathcal{A}}^{\star} (s', \varepsilon, \varepsilon)$.

Por esta razão, $x \in \mathcal{L}(\mathcal{A})$ se e só se x é gerada a partir de $[s_0, \mathtt{Z}, s_0]$ ou de $[s_0, \mathtt{Z}, s_1]$, já que, por definição de linguagem aceite pelo autómato, $x \in \mathcal{L}(\mathcal{A})$ se e só se $(s_0, x, \mathtt{Z}) \vdash_{\mathcal{A}}^{\star} (s_0, \varepsilon, \varepsilon)$ ou $(s_0, x, \mathtt{Z}) \vdash_{\mathcal{A}}^{\star} (s_1, \varepsilon, \varepsilon)$. Assim, o algoritmo define como S-produções as seguintes:

$$\begin{array}{lcl} S & \to & [s_0, \mathtt{Z}, s_0] \\ S & \to & [s_0, \mathtt{Z}, s_1] \end{array}$$

Para as restantes transições tem-se:

- $\delta(s_0, \varepsilon, \mathtt{Z}) = \{(s_1, \mathtt{Z})\}$: sabemos que, estando no estado s_0 e tendo Z no topo da pilha, o autómato consegue retirar Z quando consome um dado x, se conseguir retirar Z com x a partir do estado s_1. O estado a que chega após consumir x poderá ser s_0 ou s_1. Assim, $\delta(s_0, \varepsilon, \mathtt{Z}) = \{(s_1, \mathtt{Z})\}$ é traduzida por duas regras de produção:

$$\begin{aligned} [s_0, \mathtt{Z}, s_0] &\rightarrow [s_1, \mathtt{Z}, s_0] \\ [s_0, \mathtt{Z}, s_1] &\rightarrow [s_1, \mathtt{Z}, s_1] \end{aligned}$$

- $\delta(s_0, \mathtt{0}, \mathtt{Z}) = \{(s_0, \mathtt{AZ})\}$ dá origem às quatro regras seguintes, que resultam de o estado final poder ser s_1 ou s_0 e o intermédio também:

$$\begin{aligned} [s_0, \mathtt{Z}, s_1] &\rightarrow \mathtt{0}[s_0, \mathtt{A}, s_0][s_0, \mathtt{Z}, s_1] & \qquad [s_0, \mathtt{Z}, s_0] &\rightarrow \mathtt{0}[s_0, \mathtt{A}, s_0][s_0, \mathtt{Z}, s_0] \\ [s_0, \mathtt{Z}, s_1] &\rightarrow \mathtt{0}[s_0, \mathtt{Z}, s_1][s_1, \mathtt{Z}, s_1] & \qquad [s_0, \mathtt{Z}, s_0] &\rightarrow \mathtt{0}[s_0, \mathtt{A}, s_1][s_1, \mathtt{Z}, s_0] \end{aligned}$$

  $[s_0, \mathtt{A}, s]$ denota o conjunto das palavras que, estando o autómato no estado s_0 e tendo A no topo da pilha, retiram A levando o autómato ao estado s, para $s \in \{s_0, s_1\}$. Ou seja, tem-se $(s_0, x_1, \mathtt{A}\alpha) \vdash_{\mathcal{A}}^{\star} (s, \varepsilon, \alpha)$ para algum $s \in \{s_0, s_1\}$,

  Notamos que, se para $x_1 \in \{\mathtt{0}, \mathtt{1}\}^{\star}$ se tem $(s_0, x_1, \mathtt{A}\alpha) \vdash_{\mathcal{A}}^{\star} (s, \varepsilon, \alpha)$ para algum $s \in \{s_0, s_1\}$, ou seja, estando $\mathcal{A}$ no estado s_0 e tendo A no topo da pilha, a palavra x_1 retira A e leva o autómato ao estado s. Suponhamos agora que para um dado $x_2 \in \{\mathtt{0}, \mathtt{1}\}^{\star}$ se tem $(s, x_2, \mathtt{Z}\beta) \vdash_{\mathcal{A}}^{\star} (s', \varepsilon, \beta)$ para algum $s' \in \{s_0, s_1\}$. Então, no estado s_0 com Z no topo da pilha, a palavra $\mathtt{0}x_1x_2$ leva o autómato ao estado s' retirando esse Z da pilha, pois

$$(s_0, \mathtt{0}x_1x_2, Z\alpha) \vdash_{\mathcal{A}} (s_0, x_1x_2, \mathtt{AZ}\alpha) \vdash_{\mathcal{A}}^{\star} (s, x_2, \mathtt{Z}\alpha) \vdash_{\mathcal{A}}^{\star} (s', \varepsilon, \alpha)$$

  Note que, se não analisarmos o comportamento do autómato com algum cuidado, podemos apenas dizer que depois de conseguir retirar o símbolo A que colocou na pilha, o autómato pode estar quer em s_0 quer em s_1.

- Analisando as restantes transições, obtemos ainda as regras seguintes.

$$\begin{aligned} [s_0, \mathtt{A}, s_0] &\rightarrow \mathtt{0}[s_0, \mathtt{A}, s_0][s_0, \mathtt{A}, s_0] \\ [s_0, \mathtt{A}, s_0] &\rightarrow \mathtt{0}[s_0, \mathtt{A}, s_1][s_1, \mathtt{A}, s_0] \\ [s_0, \mathtt{A}, s_1] &\rightarrow \mathtt{0}[s_0, \mathtt{A}, s_0][s_0, \mathtt{A}, s_1] \\ [s_0, \mathtt{A}, s_1] &\rightarrow \mathtt{0}[s_0, \mathtt{A}, s_1][s_1, \mathtt{A}, s_1] \\ [s_0, \mathtt{A}, s_1] &\rightarrow \mathtt{1} \\ [s_1, \mathtt{A}, s_1] &\rightarrow \mathtt{1} \\ [s_1, \mathtt{Z}, s_1] &\rightarrow \mathtt{1} \\ [s_1, \mathtt{Z}, s_1] &\rightarrow \mathtt{1}[s_1, \mathtt{Z}, s_1] \\ [s_1, \mathtt{Z}, s_0] &\rightarrow \mathtt{1}[s_1, \mathtt{Z}, s_0] \end{aligned}$$

Se substituirmos os nomes das variáveis, obtemos as regras seguintes (na ordem em que foram sendo apresentadas):

$$\begin{aligned}
S &\rightarrow C \mid D \\
C &\rightarrow E \mid \mathtt{0}MC \mid \mathtt{0}RE \\
D &\rightarrow F \mid \mathtt{0}MD \mid \mathtt{0}RF \\
M &\rightarrow \mathtt{0}MM \mid \mathtt{0}RT \mid \mathtt{0}MR \mid \mathtt{0}RW \mid \mathtt{1} \\
W &\rightarrow \mathtt{1} \\
F &\rightarrow \mathtt{1} \mid \mathtt{1}F \\
E &\rightarrow \mathtt{1}E
\end{aligned}$$

É evidente que E não gera qualquer palavra de $\{\mathtt{0},\mathtt{1}\}^\star$. Portanto, E é desnecessária, e pode ser retirada. Por outro lado, não há qualquer regra que defina T. Assim, podemos eliminar também as regras em que T ocorre.

$$\begin{aligned}
S &\rightarrow C \mid D \\
C &\rightarrow \mathtt{0}MC \\
D &\rightarrow F \mid \mathtt{0}MD \mid \mathtt{0}RF \\
M &\rightarrow \mathtt{0}MM \\
R &\rightarrow \mathtt{0}MR \mid \mathtt{0}RW \mid \mathtt{1} \\
W &\rightarrow \mathtt{1} \\
F &\rightarrow \mathtt{1} \mid \mathtt{1}F
\end{aligned}$$

Verificamos agora que C e M são dispensáveis, já que não geram quaisquer palavras. Por outro lado, como W só gera $\mathtt{1}$, podemos substituir todas as ocorrências de W por $\mathtt{1}$. Simplificando, vem:

$$\begin{aligned}
S &\rightarrow D \\
D &\rightarrow F \mid \mathtt{0}RF \\
R &\rightarrow \mathtt{0}R\mathtt{1} \mid \mathtt{1} \\
F &\rightarrow \mathtt{1} \mid \mathtt{1}F
\end{aligned}$$

Ainda podemos eliminar D, pois vemos que S e D geram as mesmas palavras. Finalmente, obtém-se

$$\begin{aligned}
S &\rightarrow F \mid \mathtt{0}RF \\
R &\rightarrow \mathtt{0}R\mathtt{1} \mid \mathtt{1} \\
F &\rightarrow \mathtt{1} \mid \mathtt{1}F
\end{aligned}$$

É interessante observar que, atendendo à substituição de nomes efetuada, as regras que finalmente obtivemos são:

$$\begin{array}{rcl} S & \to & [s_1,\mathtt{Z},s_1] \quad | \quad \mathtt{0}[s_0,\mathtt{A},s_1][s_1,\mathtt{Z},s_1] \\ [s_0,\mathtt{A},s_1] & \to & \mathtt{0}[s_0,\mathtt{A},s_1]\mathtt{1} \quad | \quad \mathtt{1} \\ [s_1,\mathtt{Z},s_1] & \to & \mathtt{1} \quad | \quad \mathtt{1}[s_1,\mathtt{Z},s_1] \end{array}$$

Estas regras estão mais de acordo com o comportamento do autómato $\mathcal{A}$. De facto, depois de $\mathcal{A}$ analisar palavras da linguagem está sempre no estado s_1. E, após retirar um dado $\mathtt{A}$ da pilha, $\mathcal{A}$ está necessariamente em s_1.

6.3 Simplificação de gramáticas e formas normais

Duas gramáticas $\mathcal{G}$ e $\mathcal{G}'$ são equivalentes se e só se $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$. Nesta secção, vamos ver que qualquer GIC $\mathcal{G} = (V, \Sigma, P, S)$, com $\mathcal{L}(\mathcal{G}) \neq \emptyset$, pode ser transformada numa GIC equivalente $\mathcal{G}' = (V', \Sigma, P', S)$ tal que:

- cada símbolo em Σ ocorre em alguma palavra em $\mathcal{L}(\mathcal{G}')$,
- cada variável em V ocorre em alguma derivação de alguma palavra em $\mathcal{L}(\mathcal{G}')$,
- nenhuma variável produz ε, com possível excepção de S, e
- $\mathcal{G}'$ não inclui *regras unitárias*, isto é, regras da forma $A \to B$, quaisquer que sejam as variáveis A e B.

O processo de transformação resulta dos Lemas 9–13, enunciados abaixo. Omitimos alguns detalhes das provas, que podem ser encontrados, por exemplo, em [Hopcroft & Ullman].

6.3.1 Eliminar variáveis que não geram sequências de terminais

Lema 9 *Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma GIC tal que $\mathcal{L}(\mathcal{G}) \neq \emptyset$. Existe um algoritmo que obtém uma GIC $\mathcal{G}' = (V', \Sigma, P', S)$ equivalente a $\mathcal{G}$, onde cada variável $A \in V' \subseteq V$ gera alguma sequência em $\Sigma^\star$ (ou seja, em $\mathcal{G}'$, para cada $A \in V'$, existe $w \in \Sigma^\star$ tal que $A \Rightarrow_{\mathcal{G}}^\star w$).*

Ideia da prova: A ideia é começar por definir V' como o conjunto de variáveis que trivialmente produzem sequências de terminais (por terem regras que só têm sequências de terminais no lado direito). Em seguida, completa-se V', acrescentando variáveis que têm produções que, no lado direito, incluem apenas variáveis que já estão em V', e sucessivamente.

Assim, para obter V' e P', começa-se por definir W como $\emptyset$ e V' como $\{A \mid (A \to \alpha) \in P, \text{ para algum } \alpha \in \Sigma^\star\}$. Depois, enquanto $W \neq V'$, substitui-se W por V' e V' por $W \cup \{A \mid (A \to \alpha) \in P, \text{ para algum } \alpha \in (\Sigma \cup W)^\star\}$. Quando $W = V'$, isto é, quando V' se torna invariante, define-se P' como o subconjunto das regras de P em que não ocorrem variáveis que não estão em V', ou seja, $\{A \to \alpha \mid A \in V',\ (A \to \alpha) \in P,\ \ \alpha \in (\Sigma \cup V')^\star\}$. □

Decidir se uma gramática gera a linguagem vazia. O algoritmo descrito na prova do Lema 9 pode ser aplicado a uma qualquer gramática $\mathcal{G} = (V, \Sigma, P, S)$ para verificar se $\mathcal{L}(\mathcal{G}) = \emptyset$, ou não. De facto, $\mathcal{G}$ não gera qualquer palavra se e só se S não pertencer ao conjunto das variáveis da gramática $\mathcal{G}'$ obtida pelo algoritmo, isto é, a V'.

Exemplo 117 Seja $\mathcal{G}$ a gramática dada por $\mathcal{G} = (\{S, C, D, M, R, Z, F, E, T\}, \{\mathtt{0}, \mathtt{1}\}, P, S)$, onde P é formado por:

$$\begin{array}{lcllcl} S & \to & C \mid D & & D & \to & F \mid \mathtt{0}MD \mid \mathtt{0}RF \\ C & \to & \mathtt{0}MC \mid \mathtt{0}RE \mid E & & M & \to & \mathtt{0}MM \mid \mathtt{0}RT \\ R & \to & \mathtt{0}MR \mid \mathtt{0}RZ \mid \mathtt{1} & & Z & \to & \mathtt{1} \\ F & \to & \mathtt{1} \mid \mathtt{1}F & & E & \to & \mathtt{1}E \end{array}$$

De acordo com o algoritmo descrito na prova, começamos por considerar que $W := \emptyset$ e que $V' := \{Z, R, F\}$. Como $W \neq V'$, fazemos $W := \{Z, R, F\}$ e $V' := \{Z, R, F\} \cup \{D\}$, por $(D \to F)$, por exemplo. Como $W \neq V'$, procuramos regras com D no lado direito e, possivelmente Z, R, F ou terminais. Como $(S \to D) \in P$, efetuamos as substituições, $W := \{Z, R, F, D\}$ e $V' := \{Z, R, F, D\} \cup \{S\}$. Como ainda se tem $W \neq V'$, substituimos novamente, obtendo $W := \{Z, R, F, D, S\}$ e $V' := \{Z, R, F, D, S\} \cup \{\ \}$, pois não há regras com S no lado direito. Finalmente, W e V' são iguais.

Como resultado, obtemos a gramática $\mathcal{G}' = (\{Z, R, F, D, S\}, \{\mathtt{0}, \mathtt{1}\}, P', S)$, com P' constituído pelas produções:

$$\begin{array}{lcllcl} S & \to & D & & D & \to & F \mid \mathtt{0}RF \\ R & \to & \mathtt{0}RZ \mid \mathtt{1} & & Z & \to & \mathtt{1} \\ F & \to & \mathtt{1} \mid \mathtt{1}F & & & & \end{array}$$

6.3.2 Eliminar variáveis que não ocorrem em derivações a partir do símbolo inicial

Sendo $\mathcal{L}(\mathcal{G}) \neq \emptyset$, dizemos que um **símbolo** de $V \cup \Sigma$ é **desnecessário** se não ocorrer em nenhuma derivação de palavras de $\mathcal{L}(\mathcal{G})$. Uma variável é desnecessária se não produzir sequências de terminais (isto é, se a linguagem gerada a partir dessa variável for vazia) ou se não ocorrer na derivações a partir de S. Por aplicação do algoritmo descrito na prova do Lema 9, podemos obter uma gramática que ainda contém variáveis ou terminais desnecessários, embora todas as variáveis produzam linguagens não vazias. De facto, pode acontecer que $\mathcal{G}'$ inclua símbolos que não ocorrem em qualquer derivação a partir de S. Podemos continuar a simplificar a gramática, se removermos esses símbolos, por aplicação do algoritmo descrito na prova do Lema 10.

Lema 10 *Dada uma gramática independente de contexto $\mathcal{G} = (V, \Sigma, P, S)$, existe um algoritmo que determina uma gramática independente de contexto $\mathcal{G}' = (V', \Sigma', P', S)$ equivalente a $\mathcal{G}$, em que cada variável $A \in V' \subseteq V$ ocorre em alguma derivação a partir de S (ou seja, $S \Rightarrow^\star_{\mathcal{G}'} w_1 A w_2$, para algum w_1 e algum w_2 em $(V' \cup \Sigma')^\star$).*

Ideia da prova: Começar por definir $V' := \{S\}$, $\Sigma' := \emptyset$ e $P' := \emptyset$. Acrescentar a V' todas as variáveis que ocorrem em S-produções, a Σ' todos os terminais que ocorrem em S-produções, e a P' todas as S-produções. Proceder de forma análoga para todas as outras variáveis que estiverem (ou forem sendo colocadas) em V'. □

Exemplo 118 Considerar a gramática $\mathcal{G} = (\{A, B, C, D\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, P, A)$ cujas produções são

$$\begin{array}{lcllcl} A & \to & \mathtt{a}\mathtt{a}D\mathtt{b} \mid B & \qquad B & \to & \mathtt{b} \mid \mathtt{b}\mathtt{a}B \\ D & \to & \varepsilon \mid \mathtt{a}\mathtt{a}D\mathtt{b} & \qquad C & \to & \mathtt{c}CD \mid \mathtt{c} \end{array}$$

Se aplicarmos a $\mathcal{G}$ o algoritmo dado na prova do Lema 9, não conseguimos efetuar qualquer simplificação. Apliquemos agora o algoritmo dado na prova do Lema 10, para eliminar as variáveis e terminais que não ocorram em qualquer derivação a partir de A (símbolo inicial de $\mathcal{G}$). Começamos por tomar $V' := \{A\}$, $\Sigma' := \{\ \}$ e $P' := \{\ \}$. Juntamos a P' todas as A-produções, ou seja $P' := \{A \to \mathtt{a}\mathtt{a}D\mathtt{b}, A \to B\}$, a V' as variáveis B e D, e a Σ' os símbolos $\mathtt{a}$ e $\mathtt{b}$.

Como acrescentámos as variáveis B e D a V', juntamos a P' todas as B-produções e D-produções. Como tal não traz novas entradas para V' nem para Σ', o processo termina, resultando $\mathcal{G}' = (\{A, B, D\}, \{\mathtt{a}, \mathtt{b}\}, P', A)$, com

$$P' = \{A \to \mathtt{a}\mathtt{a}D\mathtt{b},\ A \to B,\ B \to \mathtt{b},\ B \to \mathtt{b}\mathtt{a}B,\ D \to \varepsilon,\ D \to \mathtt{a}\mathtt{a}D\mathtt{b}\}.$$

Lema 11 *A gramática que se obtém de uma GIC $\mathcal{G}$, tal que $\mathcal{L}(\mathcal{G}) \neq \emptyset$, se se aplicar o algoritmo descrito na prova do Lema 9 e, em seguida, o algoritmo descrito na prova do Lema 10, não tem qualquer símbolo desnecessário.*

6.3.3 Garantir que nenhuma variável distinta do símbolo inicial gera a palavra vazia

Lema 12 *Existe um algoritmo que, dada uma gramática independente de contexto $\mathcal{G} = (V, \Sigma, P, S)$, determina uma gramática $\mathcal{G}_1 = (V_1, \Sigma, P_1, S)$, que gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$, tal que $\mathcal{G}_1$ não tem símbolos desnecessários e nenhuma das variáveis em V_1 produz ε (ou seja, $A \not\Rightarrow^{\star}_{\mathcal{G}_1} \varepsilon$, para todo $A \in V_1$).*

Ideia da prova: A técnica é semelhante à usada nas provas dos Lemas 9 e 10. Começamos por determinar todas as variáveis em V que geram ε. Para isso, começamos por marcar todas as variáveis A tais que $(A \to \varepsilon) \in P$, como *"variáveis que geram ε"*. Recursivamente, marcamos todas as variáveis A tais que $(A \to \alpha) \in P$ e α é uma sequência de variáveis já marcadas (ou seja, variáveis que já se sabe que geram ε). O processo é sucessivamente repetido até não ser possível marcar mais variáveis.

Por fim, para determinar o conjunto P' (que poderá incluir produções que não estão em P), aplica-se o método seguinte. Para cada uma das regras $(A \to X_1 X_2 \cdots X_n) \in P$, com $X_i \in V \cup \Sigma$, colocar em P' todas as produções $A \to \alpha_1 \alpha_2 \cdots \alpha_n$ obtidas dessa regra e que satisfazem as condições seguintes:

- se X_i for um terminal ou uma variável não marcada, então $\alpha_i = X_i$;
- se $X_i \in V$ estiver marcada, então α_i pode ser ε ou X_i.
- os α_i's não podem ser todos ε.

A gramática $\mathcal{G}' = (V, \Sigma, P', S)$ gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$ e não tem variáveis que gerem ε. Para obter uma gramática equivalente mas que não tenha símbolos desnecessários, aplica-se a $\mathcal{G}'$ o algoritmo dado na prova do Lema 9 e em seguida o dado na prova do Lema 10. □

Exemplo 119 Vamos aplicar o algoritmo dado na prova do Lema 12 para obter uma gramática G_1 que gere $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$ e não tenha variáveis que gerem ε, sendo $\mathcal{G}$ dada por

$$\mathcal{G} = (\{S, T, C\}, \{\mathtt{a}, \mathtt{b}\}, \{T \to \varepsilon,\ T \to \mathtt{a}T,\ S \to T,\ S \to \mathtt{a}S\mathtt{b},\ C \to ST,\ C \to \mathtt{ab}\}, S).$$

Como $(T \to \varepsilon) \in P$, marcamos T como variável que gera ε. Consequentemente, como $(S \to T) \in P$, também S é marcado. Como S e T estão marcados e $(C \to ST) \in P$, também C é marcado.

Determinamos em seguida a gramática $\mathcal{G}' = (V, \Sigma, P', S)$. A produção $T \to \varepsilon$ de $\mathcal{G}$ não é incluída em P'. A produção $T \to \mathtt{a}T$ é substituída por duas produções $T \to \mathtt{a}T$ e $T \to \mathtt{a}$. Analogamente, incluimos $S \to T$ em P', mas não a regra $S \to \varepsilon$ resultante da substituição de T por ε (como referido na prova do lema). Aplicando o mesmo procedimento às restantes regras de S e C, obtém-se $\mathcal{G}' = (\{S, T, C\}, \{\mathtt{a}, \mathtt{b}\}, P', S)$, onde P' é constituído por:

$$\begin{aligned} T &\to \mathtt{a} \mid \mathtt{a}T \\ S &\to T \mid \mathtt{a}S\mathtt{b} \mid \mathtt{ab} \\ C &\to S \mid T \mid ST \mid \mathtt{ab} \end{aligned}$$

A gramática $\mathcal{G}'$ gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$ e não tem variáveis que gerem ε. Notamos que todas as variáveis geram sequências de terminais. Assim, se aplicarmos a $\mathcal{G}'$ o algoritmo dado no Lema 9, não conseguimos efetuar qualquer simplificação. No entanto, há variáveis (mais precisamente C) que não ocorrem em derivações a partir de S. Consequentemente, se aplicarmos o algoritmo descrito no Lema 10, conseguimos eliminar C e obter uma gramática que gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$ e não tem símbolos que gerem ε nem símbolos que não ocorrem na derivação de alguma palavra dessa linguagem:

$$(\{S, T\}, \{\mathtt{a}, \mathtt{b}\}, \{T \to \mathtt{a},\ T \to \mathtt{a}T,\ S \to T,\ S \to \mathtt{a}S\mathtt{b},\ S \to \mathtt{ab}\}).$$

6.3.4 Eliminar regras unitárias

Designa-se por **produção unitária** qualquer regra da forma $A \to B$, com A e B variáveis.

Lema 13 *Qualquer LIC L, tal que $L \neq \emptyset$ e $\varepsilon \notin L$, pode ser ser gerada por uma GIC que não tem produções unitárias, nem símbolos que não ocorrem na derivação de palavras de L, nem variáveis que gerem ε.*

Ideia da prova: Pelo Lema 12, existe uma gramática $\mathcal{G} = (V, \Sigma, P, S)$ que gera L e que não tem nem variáveis que gerem ε (nem símbolos desnecessários). Se $\mathcal{G}$ não tiver produções unitárias, então nada resta fazer. Caso contrário, vamos definir um novo conjunto de produções P' com base em P, no qual não há produções unitárias, sendo a gramática resultante equivalente a $\mathcal{G}$.

Como $\mathcal{G}$ não tem regras da forma $A \to \varepsilon$, tem-se $A \Rightarrow_{\mathcal{G}}^{\star} B$ se e só se $A \Rightarrow_{\mathcal{G}} B_1 \Rightarrow_{\mathcal{G}} \cdots \Rightarrow_{\mathcal{G}} B_n \Rightarrow_{\mathcal{G}} B$, para algum $n \geq 0$, tal que as $B_i \neq B_j$ se $i \neq j$. Sem perda de generalidade, podemos considerar que as variáveis são todas distintas (se não, poderiamos considerar uma derivação mais curta). Notamos que, se $n = 0$, estamos a considerar simplesmente $A \Rightarrow_{\mathcal{G}} B$.

Começamos por tomar P' como $P \setminus \{A \to B \mid A, B \in V, (A \to B) \in P\}$, ou seja, como o conjunto de todas as produções em P que não são unitárias. Depois, para cada variável A, se se tiver $A \Rightarrow_{\mathcal{G}}^{\star} B$ para alguma variável B diferente de A, então colocamos em P' todas as produções da forma $A \to \alpha$, tais que $(B \to \alpha) \in P$ e α não é uma variável.

Finalmente, simplificamos a gramática resultante, usando o algoritmo dado na prova Lema 9 e, em seguida, o algoritmo dado na prova do Lema 10. □

Em alternativa, pode-se aplicar o método seguinte para eliminar as regras unitárias. Enquanto houver regras unitárias, tomar uma qualquer $A \to B$ e substitui-la pelas regras $A \to \beta$, tais que $B \to \beta$ e $\beta \neq A$. Pode acontecer que nesta substituição se criem novas regras unitárias, mas o processo terminará.

Exemplo 120 Seja $\mathcal{G} = (\{S, T\}, \{\mathtt{a}, \mathtt{b}\}, \{T \to \mathtt{a},\ T \to \mathtt{a}T,\ S \to T,\ S \to \mathtt{a}S\mathtt{b},\ S \to \mathtt{ab}\})$. Segundo a prova do Lema 13, podemos retirar a produção $S \to T$, se a substituirmos pelas produções $S \to \mathtt{a}$ e $S \to \mathtt{a}T$. Obtemos a gramática $(\{S, T\}, \{\mathtt{a}, \mathtt{b}\}, \{T \to \mathtt{a},\ T \to \mathtt{a}T,\ S \to \mathtt{a},\ S \to \mathtt{a}T,\ S \to \mathtt{a}S\mathtt{b},\ S \to \mathtt{ab}\})$, que não tem nem variáveis desnecessárias, nem regras unitárias nem da forma $A \to \varepsilon$.

Exemplo 121 Seja $\mathcal{G} = (\{S, A, B, C, D\}, \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}, P, S)$ com P dado por:

$$\begin{aligned} S &\to \mathtt{0}S \mid B \\ B &\to \mathtt{1}B \mid C \\ C &\to D \mid C\mathtt{2} \mid \mathtt{2} \mid S \\ D &\to \mathtt{0}D\mathtt{1} \mid \mathtt{01} \end{aligned}$$

Para transformar $\mathcal{G}$ numa gramática equivalente mas que não tem regras unitárias, podemos:

- retirar $C \to D$ e acrescentar $C \to \mathtt{0}D\mathtt{1} \mid \mathtt{01}$;
- retirar $C \to S$ e acrescentar $C \to \mathtt{0}S \mid B$;
- retirar $C \to B$, e acrescentar $C \to \mathtt{1}B$ (mas não $C \to C$);
- retirar $B \to C$, e acrescentar $B \to \mathtt{0}D\mathtt{1} \mid \mathtt{01} \mid C\mathtt{2} \mid \mathtt{2} \mid \mathtt{0}S \mid \mathtt{1}B$, pois, após as três primeiras alterações, as regras para C são $C \to \mathtt{0}D\mathtt{1} \mid \mathtt{01} \mid C\mathtt{2} \mid \mathtt{2} \mid \mathtt{0}S \mid \mathtt{1}B$;
- e, finalmente, eliminar $S \to B$, resultando

$$\begin{aligned} S &\to \mathtt{0}S \mid \mathtt{1}B \mid \mathtt{0}D\mathtt{1} \mid \mathtt{01} \mid C\mathtt{2} \mid \mathtt{2} \mid \mathtt{0}S \mid \mathtt{1}B \\ B &\to \mathtt{1}B \mid \mathtt{0}D\mathtt{1} \mid \mathtt{01} \mid C\mathtt{2} \mid \mathtt{2} \mid \mathtt{0}S \mid \mathtt{1}B \\ C &\to \mathtt{0}D\mathtt{1} \mid \mathtt{01} \mid C\mathtt{2} \mid \mathtt{2} \mid \mathtt{0}S \mid \mathtt{1}B \\ D &\to \mathtt{0}D\mathtt{1} \mid \mathtt{01} \end{aligned}$$

6.3.5 Redução à forma normal de Chomsky

Definição 17 *Uma gramática independente de contexto $\mathcal{G} = (V, \Sigma, P, S)$ está na* **forma normal de Chomsky** *sse qualquer produção é da forma $A \to BC$ ou $A \to a$, com $A, B, C \in V$ e $a \in \Sigma$, quaisquer.*

De acordo com esta definição, que será a que adoptaremos, se $\mathcal{G}$ estiver na forma normal de Chomsky, $\varepsilon \notin \mathcal{L}(\mathcal{G})$. Existe uma definição alternativa que permite ter $\varepsilon \in \mathcal{L}(\mathcal{G})$. Contudo, requer que S não ocorra no lado direito de nenhuma regra (isto é, $B \neq S$ e $C \neq S$ em $A \to BC$) e que nenhuma variável produza ε a não ser S (se $\mathcal{G}$ incluir a produção $S \to \varepsilon$). Qualquer linguagem independente de contexto pode ser gerada por uma GIC na forma normal de Chomsky, se se admitir esta definição. Mais adiante, voltaremos a este assunto.

Proposição 40 *Qualquer linguagem independente de contexto, que não tenha ε, é gerada por uma gramática na forma normal de Chomsky.*

Ideia da prova: Se $L = \emptyset$, então $\mathcal{G} = (\{S\}, \Sigma, \{\}, S)$ gera $\mathcal{L}$ e satisfaz as condições da definição de forma normal de Chomksy pois não tem regras. Assim, seja L uma linguagem não vazia, tal que $\varepsilon \notin L$. Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma gramática sem produções $A \to \varepsilon$ nem $A \to B$, com $A, B \in V$, a qual pode ser obtida a partir de qualquer gramática que gere L, como descrevemos anteriormente (Lemas 9–13).

Por definição de forma normal de Chomsky, qualquer regra em P que seja da forma $A \to a$, com $a \in \Sigma$, está já nas condições pretendidas. Como $\mathcal{G}$ não tem produções unitárias, as restantes regras em P devem ser da forma

$A \to X_1X_2\cdots X_n$ para algum $n \geq 2$, cada X_i podendo ser um terminal ou uma variável. Se for um terminal, seja por exemplo $X_i = a$, introduzimos uma nova variável $C_a \to a$, e substituimos todas as ocorrências de a em regras dessa forma por C_a.

Repetindo este processo, substituimos todas as regras da forma $A \to X_1X_2\cdots X_n$, com $n \geq 2$, por regras da forma $A \to Y_1Y_2\cdots Y_n$, em que $Y_i = X_i$, se X_i era variável, ou $Y_i = C_a$, para algum a, verificando-se que tais regras só ficam com variáveis. Todas as regras obtidas, que tenham três ou mais variáveis na parte direita, ainda não estão na forma pretendida. Mas cada uma delas pode ser substituída por um conjunto de regras cuja parte direita é uma sequência de duas variáveis. Essa transformação é quase trivial, bastando introduzir novas variáveis D_1,... D_{n-2} e as regras $A \to Y_1D_1$, $D_1 \to Y_2D_2$, ..., e $D_{n-2} \to Y_{n-1}Y_n$.

A gramática obtida está na forma normal de Chomsky e podemos mostrar que é equivalente a $\mathcal{G}$. □

Exemplo 122 Seja $\mathcal{G} = (\{S,T\}, \{\mathtt{a},\mathtt{b}\}, \{T \to \mathtt{a},\ T \to \mathtt{a}T,\ S \to \mathtt{a},\ S \to \mathtt{a}T,\ S \to \mathtt{a}S\mathtt{b},\ S \to \mathtt{ab}\}, S)$. Para obter uma gramática na forma normal de Chomsky que seja equivalente a $\mathcal{G}$, vamos aplicar o método que acábamos de descrever. Notamos que $\mathcal{G}$ não tem produções $V \to \varepsilon$ nem unitárias. Transformamos o conjunto de produções, substituindo por uma variável cada terminal que ocorra em regras cuja parte direita não é um único terminal. O conjunto de produções fica

$$\{T \to \mathtt{a}, T \to AT,\ A \to \mathtt{a},\ S \to \mathtt{a},\ S \to AT,\ S \to ASB,\ B \to \mathtt{b},\ S \to AB\}$$

verificando-se que a regra $S \to ASB$ não está na forma pretendida, pelo que a substituimos por duas regras $S \to AX$ e $X \to SB$. Obtém-se a gramática $\mathcal{G}' = (\{S,T,A,B,X\}, \{\mathtt{a},\mathtt{b}\}, P', S)$, que é equivalente a $\mathcal{G}$, mas está na forma normal de Chomsky, sendo P' dado por:

$$P' = \{T \to \mathtt{a}, T \to AT,\ A \to \mathtt{a},\ S \to \mathtt{a},\ S \to AT,\ S \to AX,\ X \to SB,\ B \to \mathtt{b},\ S \to AB\}.$$

Exemplo 123 Seja $\mathcal{G} = (\{E,F,T,N,D,R\}, \{\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{3},\mathtt{4},\mathtt{5},\mathtt{6},\mathtt{7},\mathtt{8},\mathtt{9},\ \mathtt{(},\mathtt{)},\mathtt{+},\mathtt{-},\mathtt{*},\mathtt{/}\}, P, E)$, com P dado por:

$$\begin{aligned}
E &\to T\mathtt{+}E \mid T\mathtt{-}E \mid T \\
T &\to F \mid F\mathtt{*}T \mid F\mathtt{/}T \\
F &\to \mathtt{(}E\mathtt{)} \mid N \\
N &\to \mathtt{0} \mid D \mid DR \\
D &\to \mathtt{1} \mid \mathtt{2} \mid \mathtt{3} \mid \mathtt{4} \mid \mathtt{5} \mid \mathtt{6} \mid \mathtt{7} \mid \mathtt{8} \mid \mathtt{9} \\
R &\to \mathtt{0} \mid D \mid \mathtt{0}R \mid DR
\end{aligned}$$

A gramática $\mathcal{G}$ gera expressões aritméticas com inteiros não negativos representados em decimal, isto é, na base 10. Vamos determinar uma gramática na forma de Chomsky equivalente a $\mathcal{G}$. Vemos que $\mathcal{G}$ que não tem variáveis que produzam ε, mas tem produções unitárias. Para as eliminar e obter $\mathcal{G}'$ sem regras unitárias, usamos o método dado na prova do Lema 13. Todas as regras não unitárias de $\mathcal{G}$ fazem parte do conjunto P' (de regras de $\mathcal{G}'$), pelo que temos

$$\begin{aligned}
E &\to T\texttt{+}E \mid T\texttt{-}E \\
T &\to F\star T \mid F/T \\
F &\to \texttt{(}E\texttt{)} \\
N &\to \texttt{0} \mid DR \\
D &\to \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
R &\to \texttt{0} \mid \texttt{0}R \mid DR
\end{aligned}$$

Como $E \Rightarrow^\star_{\mathcal{G}} T$, $E \Rightarrow^\star_{\mathcal{G}} F$, $E \Rightarrow^\star_{\mathcal{G}} N$, e $T \Rightarrow^\star_{\mathcal{G}} D$, devemos introduzir em P' ainda

$$E \to F\star T \mid F/T \mid \texttt{(}E\texttt{)} \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

e como $T \Rightarrow^\star_{\mathcal{G}} F$, $T \Rightarrow^\star_{\mathcal{G}} N$ e $T \Rightarrow^\star_{\mathcal{G}} D$, introduzimos em P'

$$T \to \texttt{(}E\texttt{)} \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

e por $F \Rightarrow^\star_{\mathcal{G}} N$ e $F \Rightarrow^\star_{\mathcal{G}} D$, introduzimos em P' as regras

$$F \to \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

e por $N \Rightarrow^\star_{\mathcal{G}} D$, introduzimos ainda

$$N \to \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

e finalmente, por $R \Rightarrow^\star_{\mathcal{G}} D$, acrescentamos

$$R \to \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

A seguir, para determinar uma gramática equivalente mas que esteja na forma de Chomsky, aplicamos o algoritmo descrito na prova da Proposição 40. Não alteramos as regras cuja parte direita seja um único terminal. Nas restantes, substituimos cada terminal por uma nova variável.

$$\begin{aligned}
M &\to \texttt{+} \\
S &\to \texttt{-} \\
X &\to \star
\end{aligned}$$

$$
\begin{aligned}
Q &\to \texttt{/} \\
A &\to \texttt{(} \\
B &\to \texttt{)} \\
O &\to \texttt{0} \\
E &\to TME \mid TSE \mid FXT \mid FQT \mid AEB \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
T &\to FXT \mid FQT \mid AEB \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
F &\to AEB \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
N &\to \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
D &\to \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
R &\to \texttt{0} \mid OR \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}
\end{aligned}
$$

Para chegar à forma de Chomsky, resta substituir as regras cujo lado direito tem três ou mais variáveis. Substituimos $E \to TME \mid TSE \mid FXT \mid FQT \mid AEB$ por

$$
\begin{aligned}
E &\to TE_1 \mid TE_2 \mid FT_1 \mid FT_2 \mid AT_3 \\
E_1 &\to ME \\
E_2 &\to SE \\
T_1 &\to XT \\
T_2 &\to QT \\
T_3 &\to EB
\end{aligned}
$$

e, para obter uma gramática mais simples, aproveitando tal definição, substituimos $T \to FXT \mid FQT \mid AEB$ por

$$T \to FT_1 \mid FT_2 \mid AT_3$$

e, finalmente, para $F \to AEB$, em vez de $F \to AF_1$ e $F_1 \to EB$, usamos

$$F \to AT_3$$

Forma normal de Chomsky estendida (caso $\varepsilon \in \mathcal{L}(\mathcal{G})$). Como $\varepsilon \in \mathcal{L}(G)$ sse S pertence ao conjunto das variáveis de $\mathcal{G}$ que produzem ε, podemos decidir se $\varepsilon \in \mathcal{L}(\mathcal{G})$ por aplicação do método descrito na prova do Lema 12. Por outro lado, essa prova descreve um método para obter uma gramática $\mathcal{G}_1 = (V_1, \Sigma, P_1, S)$, que gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$ e em que nenhuma das variáveis em V_1 produz ε. Por conversão de $\mathcal{G}_1$ à forma normal de Chomsky, obtemos uma gramática $\mathcal{G}_1' = (V_1', \Sigma, P_1', S)$, que podemos usar para obter uma gramática $\mathcal{G}'$ equivalente a $\mathcal{G}$ e que está na forma normal de Chomsky estendida: $\mathcal{G}' = (V_1' \cup \{S_0\}, \Sigma, P_1' \cup \{S_0 \to \varepsilon\} \cup \{(S_0 \to \alpha) \mid (S \to \alpha) \in P_1'\}, S_0)$.

6.3.6 Redução à forma normal de Greibach

Como vimos, uma gramática independente de contexto $\mathcal{G} = (V, \Sigma, P, S)$ está na **forma normal de Greibach** sse qualquer produção em P é da forma $A \to aw$, com $A \in V$, $a \in \Sigma$ e $w \in V^\star$, quaisquer. Na prova da Proposição 41, apresentamos um método genérico para converter uma gramática da forma normal de Chomsky para a forma normal de Greibach. Os dois lemas seguintes são usados nessaa prova.

Lema 14 *Seja $\mathcal{G} = (V, \Sigma, P, S)$ uma gramática independente de contexto, seja $A \to \alpha_1 B \alpha_2$ uma produção de P e sejam $B \to \beta_1 \mid \cdots \mid \beta_r$ as B-produções. Obtém-se uma gramática equivalente se se substituir $A \to \alpha_1 B \alpha_2$ em $\mathcal{G}$, por $A \to \alpha_1\beta_1\alpha_2 \mid \cdots \mid \alpha_1\beta_r\alpha_2$.*

Ideia da prova: Trivial (segue imediatamente da noção de derivação). □

Lema 15 *Sejam $\mathcal{G} = (V, \Sigma, P, S)$ uma gramática independente de contexto e $A \to A\alpha_1 \mid \cdots \mid A\alpha_r$ as A-produções cujo símbolo mais à esquerda na parte direita é A. Sejam $A \to \beta_1 \mid \cdots \mid \beta_s$ as restantes A-produções em P. Seja $B \notin V$ uma variável. Obtém-se uma gramática $\mathcal{G}_1 = (V \cup \{B\}, \Sigma, P_1, S)$ equivalente a $\mathcal{G}$ se se substituir as A-produções por $A \to \beta_i \mid \beta_i B$ e $B \to \alpha_j \mid \alpha_j B$, para $1 \leq i \leq s$ e $1 \leq j \leq r$.*

Ideia da prova: (Notar que os β_i's e α_j's podem ter variáveis.) Em $\mathcal{G}$, a variável A gera $\{\beta_1, \ldots, \beta_s\}\{\alpha_1, \ldots, \alpha_r\}^\star$. Em $\mathcal{G}_1$, a variável B gera $\{\alpha_1, \ldots, \alpha_r\}\{\alpha_1, \ldots, \alpha_r\}^\star$. Logo, a variável A gera $\{\beta_1, \ldots, \beta_s\}\{\alpha_1, \ldots, \alpha_r\}^\star$. □

Proposição 41 *Qualquer linguagem independente de contexto que não tenha ε pode ser gerada por uma gramática na forma normal de Greibach.*

Ideia da prova: Já mostrámos que se $L \neq \emptyset$ e $\varepsilon \notin L$, então existe uma gramática $\mathcal{G} = (V, \Sigma, P, S)$ na forma de Chomsky que gera a L (tendo estudado também um algoritmo para converter uma qualquer gramática à forma normal de Chomsky). Vamos agora descrever um algoritmo para converter $\mathcal{G}$ da forma de Chomsky à forma de Greibach.

Suponhamos que ordenamos as variáveis de $\mathcal{G}$, passando a referir V como $V = \{A_1, A_2, \ldots, A_m\}$. Para obter a forma de Greibach, começamos por transformar as produções em P de modo que se $A_i \to A_j\gamma$ então $j > i$.

Para isso, aplicamos o método seguinte a cada uma das variáveis, começando em A_1 e prosseguindo, segundo a ordem dada acima, até A_m. É importante notar que, quando estamos a transformar as regras para A_k, já efectuámos modificações, em passos anteriores, que garantem que, para todo i com $1 \leq i < k$, se tem regras da forma $A_i \to A_j\gamma$ apenas se $j > i$.

Passamos então a descrever como se transformam as regras para A_k. Se $A_k \to A_j\gamma$ for uma regra e $j < k$, substituimo-la pelo conjunto de regras que resultam da substituição de A_j pela parte direita de cada regra para A_j, de

acordo com o Lema 14. Aplicando o mesmo processo, se necessário, a cada uma das regras assim obtidas, obtém-se um conjunto de produções da forma $A_k \to A_j\gamma$, em que $j \geq k$. O número de vezes que pode ser necessário repetir o processo não excede $k-1$.

Como não queremos ter produções da forma $A_k \to A_k\gamma$, com recursão à esquerda, podemos substitui-las como indica o Lema 15, introduzindo uma nova variável B_k. Depois de aplicarmos o método a todas as variáveis originais, só temos produções da forma:

$$\begin{aligned} A_i &\to A_j\gamma, \text{ com } j > i \\ A_i &\to a\gamma, \text{ com } a \in \Sigma \\ B_i &\to \gamma, \text{ com } \gamma \in (V \cup \{B_1, \ldots, B_{i-1}\})^\star \end{aligned}$$

Na parte direita de cada regra para A_m, o símbolo mais à esquerda tem que ser um terminal (já que não há nenhuma variável indexada com um índice superior a m). Do mesmo modo, na parte direita de cada regra para A_{m-1}, o símbolo mais à esquerda tem que ser ou A_m ou um terminal. Como o objectivo é dar à gramática a forma de Greibach, tal símbolo tem que ser um terminal. Assim, quando uma dada regra para A_{m-1} tiver A_m como símbolo mais à esquerda (na parte direita), podemos usar as produções de A_m para substituir as regras de A_{m-1} de forma a ter sempre um terminal como símbolo mais à esquerda.

Se continuarmos a aplicar o mesmo processo às restantes variáveis, tomando sucessivamente $A_{m-2}, \ldots A_2, A_1$, garantimos que o lado direito das produções dos A_i's começa por um terminal.

Finalmente analisamos as produções dos B_i's. Como consequência da gramática de partida estar na forma de Chomsky, o lado direito das produções dos B_i's não pode começar por um B_j, sendo ou um terminal ou um dos A_j's. Assim, basta aplicar, se necessário, uma transformação semelhante à que acabámos de descrever para alterar as produções dos B_i's de forma a começarem por um terminal. □

Exemplo 124 Vamos determinar uma gramática na forma normal de Greibach que seja equivalente à gramática cujas regras são as seguintes.

$$S \to (\;) \mid SS \mid (\; S \;)$$

Começamos por determinar uma gramática equivalente à dada mas que está na forma normal de Chomsky.

$$\begin{aligned} S &\to AB \mid SS \mid AR \\ R &\to SB \\ A &\to (\\ B &\to) \end{aligned}$$

Consideremos as variáveis na ordem seguinte S, R, A e B, e apliquemos o algoritmo dado na prova da Proposição 41. Verificamos que a produção $S \rightarrow SS$ não é da forma $A_i \rightarrow A_j\gamma$ com $j > i$, e por isso temos que a substituir. Introduzimos uma nova variável, seja M, e substituimos as regras para S, segundo o Lema 15, por

$$\begin{array}{lcl} S & \rightarrow & AB \mid AR \mid ABM \mid ARM \\ M & \rightarrow & S \mid SM \end{array}$$

Em seguida, verificamos as produções para R, verificando também a necessidade de substituir $R \rightarrow SB$, já que S precede R na ordem dada acima. De acordo com a prova da proposição anterior, devemos substitui-la por

$$R \rightarrow ABB \mid ARB \mid ABMB \mid ARMB$$

Não havendo problemas com as regras para A e B, podemos passar à segunda fase do algoritmo – transformar as regras de forma que o lado direito comece por um terminal. As variáveis iniciais são agora consideradas pela ordem inversa B, A, R e S, e só no fim se considera M. Vem,

$$\begin{array}{lcl} B & \rightarrow &) \\ A & \rightarrow & (\\ R & \rightarrow & (BB \mid (RB \mid (BMB \mid (RMB \\ S & \rightarrow & (B \mid (R \mid (BM \mid (RM \\ M & \rightarrow & (B \mid (R \mid (BM \mid (RM \mid (BMM \mid (RMM \end{array}$$

Observemos que, se tivessemos considerado inicialmente as variáveis na ordem R, S, A e B, efetuariamos menos passos para chegar a uma gramática, na forma de Greibach, equivalente à dada.

Exemplo 125 Seja $\mathcal{G}_1 = (\{E, F, T\}, \{\mathtt{n}, \mathtt{(}, \mathtt{)}, +, -, *, /\}, P, E)$ onde P é constituído pelas regras:

$$\begin{array}{lcl} E & \rightarrow & T{+}E \mid T{-}E \mid T \\ T & \rightarrow & F \mid F{*}T \mid F/T \\ F & \rightarrow & (E) \mid \mathtt{n} \end{array}$$

Esta gramática é inspirada na gramática estudada no Exemplo 123. Para tornar esta exposição mais clara, sendo a forma normal de Greibach dessa gramática algo extensa, introduzimos um terminal `n` para substituir as representações dos inteiros. O leitor interessado, poderá obter a forma normal de Greibach para a gramática original.

A forma normal de Chomsky para $\mathcal{G}_1$ encontra-se abaixo, sendo variáveis $E, T, F, E_1, E_2, T_1, T_2, T_3, M, S, X, Q, A$ e B, e E o símbolo inicial.

$$E \rightarrow TE_1 \mid TE_2 \mid FT_1 \mid FT_2 \mid AT_3 \mid \mathtt{n}$$

$$
\begin{aligned}
T &\rightarrow \texttt{n} \mid FT_1 \mid FT_2 \mid AT_3 \\
F &\rightarrow AT_3 \mid \texttt{n} \\
E_1 &\rightarrow ME \\
E_2 &\rightarrow SE \\
T_1 &\rightarrow XT \\
T_2 &\rightarrow QT \\
T_3 &\rightarrow EB \\
M &\rightarrow \texttt{+} \\
S &\rightarrow \texttt{-} \\
X &\rightarrow \texttt{*} \\
Q &\rightarrow \texttt{/} \\
A &\rightarrow \texttt{(} \\
B &\rightarrow \texttt{)}
\end{aligned}
$$

Para chegar à forma de Greibach, podemos procurar uma ordem para as variáveis que simplifique a aplicação da parte inicial do método. Se considerarmos as variáveis pela ordem seguinte

$$T_3, E, T, F, T_1, T_2, E_1, E_2, A, B, X, M, S, Q$$

verificamos que, em todas as regras da forma $A_i \rightarrow A_j\alpha$, a variável A_j ocorre depois da variável A_i na lista dada. Podemos passar de imediato à segunda parte do algoritmo para modificar as produções de forma que o símbolo mais à esquerda, na parte direita de cada regra, seja um terminal. Obtém-se a gramática com as regras seguintes:

$$
\begin{aligned}
Q &\rightarrow \texttt{/} \\
S &\rightarrow \texttt{-} \\
M &\rightarrow \texttt{+} \\
X &\rightarrow \texttt{*} \\
B &\rightarrow \texttt{)} \\
A &\rightarrow \texttt{(} \\
E_2 &\rightarrow \texttt{-}E \\
E_1 &\rightarrow \texttt{+}E \\
T_2 &\rightarrow \texttt{/}T
\end{aligned}
$$

$$
\begin{aligned}
T_1 &\to \star T \\
F &\to (T_3 \mid \mathtt{n} \\
T &\to (T_3T_1 \mid \mathtt{n}T_1 \mid (T_3T_2 \mid \mathtt{n}T_2 \mid (T_3 \mid \mathtt{n} \\
E &\to \mathtt{n}E_1 \mid (T_3T_1E_1 \mid \mathtt{n}T_1E_1 \mid (T_3T_2E_1 \mid \mathtt{n}T_2E_1 \mid (T_3E_1 \mid \\
&\quad \mathtt{n}E_2 \mid (T_3T_1E_2 \mid \mathtt{n}T_1E_2 \mid (T_3T_2E_2 \mid \mathtt{n}T_2E_2 \mid (T_3E_2 \mid \\
&\quad (T_3T_1 \mid \mathtt{n}T_1 \mid (T_3T_2 \mid \mathtt{n}T_2 \mid (T_3 \mid \mathtt{n} \\
T_3 &\to \mathtt{n}E_1B \mid (T_3T_1E_1B \mid \mathtt{n}T_1E_1B \mid (T_3T_2E_1B \mid \mathtt{n}T_2E_1B \mid (T_3E_1B \mid \\
&\quad \mathtt{n}E_2B \mid (T_3T_1E_2B \mid \mathtt{n}T_1E_2B \mid (T_3T_2E_2B \mid \mathtt{n}T_2E_2B \mid (T_3E_2B \mid \\
&\quad (T_3T_1B \mid \mathtt{n}T_1B \mid (T_3T_2B \mid \mathtt{n}T_2B \mid (T_3B \mid \mathtt{n}B
\end{aligned}
$$

A gramática está na forma de Greibach e pode ser usada para definir trivialmente um autómato de pilha que reconhece a linguagem que a gramática $\mathcal{G}_1$ gera. Para isso, basta usar o método dado na prova da Proposição 38.

Ordem alternativa. Se na primeira parte do algoritmo tivessemos escolhido outra ordem para as variáveis, por exemplo, $E, T, T_3, F, T_1, T_2, E_1, E_2, A, B, X, M, S, Q$, então teriamos que aplicar a primeira parte do algoritmo. A produção $T_3 \to EB$ teria que ser substituída pois E precede T_3, sendo para tal usadas as regras para E. Obteriamos:

$$T_3 \to TE_1B \mid TE_2B \mid FT_1B \mid FT_2B \mid AT_3B \mid \mathtt{n}B$$

Vemos que as regras $T_3 \to TE_1B$ e $T_3 \to TE_2B$ ainda não estão na forma pretendida, pois T precede T_3. Teriam que ser substituídas, usando as regras para T.

$$
\begin{aligned}
T_3 \to\ & \mathtt{n}E_1B \mid FT_1E_1B \mid FT_2E_1B \mid AT_3E_1B \\
& \mathtt{n}E_2B \mid FT_1E_2B \mid FT_2E_2B \mid AT_3E_2B \\
& FT_1B \mid FT_2B \mid AT_3B \mid \mathtt{n}B
\end{aligned}
$$

A gramática resultante satisfazia agora as condições que permitem passar à segunda fase do algoritmo.

Por conversão da gramática apresentada na forma de Greibach, concluimos que a linguagem gerada pela gramática $\mathcal{G}_1$ é aceite por pilha vazia pelo autómato de pilha

$$\mathcal{A} = (\{q\}, \{\mathtt{n}, \mathtt{(}, \mathtt{)}, +, -, \star, /\}, \{E, T, T_3, F, T_1, T_2, E_1, E_2, A, B, X, M, S, Q\}, \delta, q, E, \emptyset)$$

em que δ é dada por:

$$
\begin{array}{lcllcl}
\delta(q,/,Q) & = & \{(q,\varepsilon)\} & \delta(q,-,S) & = & \{(q,\varepsilon)\} \\
\delta(q,+,M) & = & \{(q,\varepsilon)\} & \delta(q,\star,X) & = & \{(q,\varepsilon)\} \\
\delta(q,),A) & = & \{(q,\varepsilon)\} & \delta(q,(,B) & = & \{(q,\varepsilon)\} \\
\delta(q,-,E_2) & = & \{(q,E)\} & \delta(q,+,E_1) & = & \{(q,E)\} \\
\delta(q,/,T_2) & = & \{(q,T)\} & \delta(q,\star,T_1) & = & \{(q,T)\} \\
\delta(q,(,F) & = & \{(q,T_3)\} & \delta(q,\mathtt{n},F) & = & \{(q,\varepsilon)\} \\
\delta(q,(,T) & = & \{(q,T_3T_1),(q,T_3T_2),(q,T_3)\} & \delta(q,\mathtt{n},T) & = & \{(q,T_1),(q,T_2),(q,\varepsilon)\}
\end{array}
$$

$$
\begin{array}{lcl}
\delta(q,\mathtt{n},E) & = & \{(q,E_1),(q,T_1E_1),(q,T_2E_1),(q,E_2),(q,T_1E_2),(q,T_2E_2),(q,T_1),(q,T_2),(q,\varepsilon)\} \\
\delta(q,(,E) & = & \{(q,T_3T_1E_1),(q,T_3T_2E_1),(q,T_3E_1),(q,T_3T_1E_2),(q,T_3T_2E_2),(q,T_3E_2), \\
 & & (q,T_3T_2),(q,T_3)\} \\
\delta(q,\mathtt{n},T_3) & = & \{(q,E_1B),(q,T_1E_1B),(q,T_2E_1B),(q,E_2B),(q,T_1E_2B),(q,T_2E_2B),(q,T_1B), \\
 & & (q,T_2B),(q,B)\} \\
\delta(q,(,T_3) & = & \{(q,T_3T_1E_1B),(q,T_3T_2E_1B),(q,T_3E_1B),(q,T_3T_1E_2B),(q,T_3T_2E_2B), \\
 & & (q,T_3E_2B),(q,T_3T_1B),(q,T_3T_2B),(q,T_3B)\}
\end{array}
$$

Exemplo 126 Consideremos a gramática $\mathcal{G} = (\{S\},\{\mathtt{a},\mathtt{b}\},\{S \to \varepsilon,\ S \to \mathtt{a}S\mathtt{b}S,\ S \to \mathtt{b}S\mathtt{a}S\},S)$, a qual define todas as palavras que têm igual número de a's e b's.

Seguindo a prova da Proposição 38, vamos determinar um autómato de pilha que reconhece esta linguagem por pilha vazia. Começaremos por determinar um autómato de pilha que reconhece $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$. Para isso, vamos determinar uma gramática na forma normal de Greibach para $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$, partindo de $\mathcal{G}$. Por aplicação do algoritmo dado na prova do Lema 12, determinamos uma gramática que gere $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$ e cujas variáveis não gerem ε:

$$S \ \to \ \mathtt{a}S\mathtt{b}S \mid \mathtt{ab}S \mid \mathtt{a}S\mathtt{b} \mid \mathtt{ab} \mid \mathtt{b}S\mathtt{a}S \mid \mathtt{ba}S \mid \mathtt{b}S\mathtt{a} \mid \mathtt{ba}$$

Como não tem produções unitárias, podemos converter esta gramática à forma de Chomsky, se aplicarmos o algoritmo dado na prova da Proposição 40. Começamos por introduzir variáveis para substituir os terminais que ocorrem nas produções que têm mais do que um símbolo na parte direita, obtendo:

$$
\begin{array}{lcl}
S & \to & ASBS \mid ABS \mid ASB \mid AB \mid BSAS \mid BAS \mid BSA \mid BA \\
A & \to & \mathtt{a} \\
B & \to & \mathtt{b}
\end{array}
$$

Resta substituir as regras que têm três ou mais variáveis na parte direita. Seguindo à risca o algoritmo (isto é, sem tentar alguma análise extra de que resultasse uma gramática simples), obteriamos:

$$
\begin{aligned}
S &\rightarrow AS_1 \mid AS_3 \mid AS_4 \mid AB \mid BS_5 \mid BS_7 \mid BS_8 \mid BA \\
A &\rightarrow \mathtt{a} \\
B &\rightarrow \mathtt{b} \\
S_1 &\rightarrow SS_2 \\
S_2 &\rightarrow BS \\
S_3 &\rightarrow BS \\
S_4 &\rightarrow SB \\
S_5 &\rightarrow SS_6 \\
S_6 &\rightarrow AS \\
S_7 &\rightarrow AS \\
S_8 &\rightarrow SA
\end{aligned}
$$

Contudo, podiamos ter chegado a uma gramática com menos variáveis, se analisassemos as regras com algum cuidado. Por exemplo, a gramática $(\{S, A, B, X_1, X_2\}, \{\mathtt{a}, \mathtt{b}\}, P', S)$, em que P' é constituído pelas regras abaixo, é mais simples do que a que acabámos de determinar e está na forma de Chomsky.

$$
\begin{aligned}
S &\rightarrow X_1X_2 \mid AX_2 \mid X_1B \mid AB \mid X_2X_1 \mid BX_1 \mid X_2A \mid BA \\
X_1 &\rightarrow AS \\
X_2 &\rightarrow BS \\
A &\rightarrow \mathtt{a} \\
B &\rightarrow \mathtt{b}
\end{aligned}
$$

Para chegar à forma de Greibach, consideremos as variáveis na ordem seguinte: S, X_1, X_2, A, B. Desta forma, dispensamos a aplicação da primeira fase do método descrito na prova da Proposição 41. Passamos diretamente à substituição das variáveis que ocorrem como símbolo mais à esquerda na parte direita de algumas regras, considerando as variáveis na ordem inversa da indicada acima. Vem,

$$
\begin{aligned}
B &\rightarrow \mathtt{b} \\
A &\rightarrow \mathtt{a} \\
X_2 &\rightarrow \mathtt{b}S
\end{aligned}
$$

$$
\begin{aligned}
X_1 &\rightarrow \mathtt{a}S \\
S &\rightarrow \mathtt{a}SX_2 \mid \mathtt{a}X_2 \mid \mathtt{a}SB \mid \mathtt{a}B \mid \mathtt{b}SX_1 \mid \mathtt{b}X_1 \mid \mathtt{b}SA \mid \mathtt{b}A
\end{aligned}
$$

Tendo a gramática na forma de Greibach que gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$, é trivial determinar um autómato de pilha que aceita $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$. Pela prova da Proposição 38, um tal autómato é $\mathcal{A} = (\{q\}, \{\mathtt{a}, \mathtt{b}\}, \{S, A, B, X_2, X_1\}, \delta, q, S, \emptyset)$, com

$$
\begin{aligned}
\delta(q, \mathtt{b}, B) &= \{(q, \varepsilon)\} & \qquad \delta(q, \mathtt{a}, A) &= \{(q, \varepsilon)\} \\
\delta(q, \mathtt{b}, X_2) &= \{(q, S)\} & \qquad \delta(q, \mathtt{a}, X_1) &= \{(q, S)\} \\
\delta(q, \mathtt{a}, S) &= \{(q, SX_2), (q, X_2), (q, SB), (q, B)\} \\
\delta(q, \mathtt{b}, S) &= \{(q, SX_1), (q, X_1), (q, SA), (q, A)\}
\end{aligned}
$$

Para ter um autómato que reconheça também a sequência vazia e consequentemente, $\mathcal{L}(\mathcal{G})$, basta acrescentar:

$$\delta(q, \varepsilon, S) = \{(q, \varepsilon)\}$$

O autómato de pilha que obtivemos é claramente não determinístico. Não existe qualquer autómato de pilha determinístico que reconheça as sequências que têm igual número de a's e de b's. De facto, depois de analisar qualquer prefixo da palavra dada e que ainda seja da linguagem, o autómato tem que estar em estado de aceitação mas também tem que poder continuar a análise da palavra.

É muito importante saber que existe um algoritmo para resolver um dado problema, porque se o implementarmos numa linguagem de programação, podemos usar um computador para resolver instâncias do problema. Contudo, como os dois exemplos anteriores sugerem, na resolução de um problema "à mão", nem sempre é menos trabalhoso seguir os métodos apresentados, podendo ser mais fácil abordar diretamente a instância que se quer resolver.

Exemplo 127 Podemos verificar que o autómato $\mathcal{A}' = (\{q\}, \{\mathtt{a}, \mathtt{b}\}, \{\mathtt{Z}, \mathtt{A}, \mathtt{B}\}, \delta, q, \mathtt{Z}, \emptyset)$ é mais simples do que o que obtivemos no Exemplo 126, e reconhece a mesma linguagem, por pilha vazia.

$$
\begin{aligned}
\delta(q, \varepsilon, \mathtt{Z}) &= \{(q, \varepsilon)\} \\
\delta(q, \mathtt{a}, \mathtt{Z}) &= \{(q, \mathtt{AZ})\} & \qquad \delta(q, \mathtt{b}, \mathtt{Z}) &= \{(q, \mathtt{BZ})\} \\
\delta(q, \mathtt{a}, \mathtt{A}) &= \{(q, \mathtt{AA})\} & \qquad \delta(q, \mathtt{b}, \mathtt{B}) &= \{(q, \mathtt{BB})\} \\
\delta(q, \mathtt{b}, \mathtt{A}) &= \{(q, \varepsilon)\} & \qquad \delta(q, \mathtt{a}, \mathtt{B}) &= \{(q, \varepsilon)\}
\end{aligned}
$$

A pilha não tem simultaneamente A's e B's. Se tiver A's, o prefixo lido tem a's em excesso. Se tiver B's, então o pefixo lido tem b's em excesso. Quando tem apenas Z, o número de a's e b's lidos é igual.

A gramática que corresponde a $\mathcal{A}'$, segundo o método descrito na prova da Proposição 38, é

$$\begin{array}{lcl} S & \to & [q,\mathtt{Z},q] \\ [q,\mathtt{Z},q] & \to & \varepsilon \\ [q,\mathtt{Z},q] & \to & \mathtt{a}[q,\mathtt{A},q]\,[q,\mathtt{Z},q] \\ [q,\mathtt{Z},q] & \to & \mathtt{b}[q,\mathtt{B},q]\,[q,\mathtt{Z},q] \\ [q,\mathtt{A},q] & \to & \mathtt{a}[q,\mathtt{A},q]\,[q,\mathtt{A},q] \\ [q,\mathtt{B},q] & \to & \mathtt{b}[q,\mathtt{B},q]\,[q,\mathtt{B},q] \\ [q,\mathtt{A},q] & \to & \mathtt{b} \\ [q,\mathtt{B},q] & \to & \mathtt{a} \end{array}$$

Tal gramática é equivalente à gramática seguinte, sendo S ainda o símbolo inicial.

$$\begin{array}{lcl} S & \to & \varepsilon \mid \mathtt{a}AS \mid \mathtt{b}BS \\ A & \to & \mathtt{a}AA \mid \mathtt{b} \\ B & \to & \mathtt{b}BB \mid \mathtt{a} \end{array}$$

6.4 Algoritmo CYK (de Cocke-Younger-Kasami)

Dada uma LIC L e uma gramática $\mathcal{G}$ que a gera, é importante poder analisar $x \in \Sigma^\star$ para verificar se pertence a L e, possivelmente, determinar uma árvore de derivação para x em $\mathcal{G}$. Quando $\mathcal{G}$ está na forma normal de Chomsky, o comprimento de uma derivação de x não poderá exceder $2|x| - 1$, por cada variável gerar algum terminal. Um algoritmo "força-bruta" que enumerasse todas as derivações de comprimento inferior a $2|x|$ permitiria verificar se $x \in \mathcal{L}(\mathcal{G})$. Contudo, esse algoritmo seria exponencial e são conhecidos algoritmos polinomiais para o problema.

Dada uma gramática $\mathcal{G} = (V, \Sigma, P, S)$ na forma normal de Chomsky, o algoritmo de Cocke-Younger-Kasami, conhecido por algoritmo CYK, decide em tempo polinomial se uma palavra $x \in \Sigma^\star$ pertence a $\mathcal{L}(\mathcal{G})$. Para $\mathcal{G}$ fixa, a complexidade temporal do algoritmo é $O(|x|^2)$, ou seja, é quadrática no comprimento da palavra que se pretende analisar. Durante a aplicação do algoritmo, pode ser mantida informação adicional que permitirá indicar uma árvore de derivação para x, se $x \in \mathcal{L}(\mathcal{G})$.

Como $\mathcal{G}$ está na forma de Chomsky, para $|x| = 1$ tem-se $x \in \mathcal{L}(\mathcal{G})$ se e só se $S \to x$. Por isso, será mais interessante considerar palavras x cujo comprimento seja maior que um. Seja $x = x_1 \ldots x_n$, com $n \geq 2$, e $x_i \in \Sigma$. O algoritmo CYK decide se $x \in \mathcal{L}(\mathcal{G})$ através da análise de todas as subsequências de x, usando programação dinâmica para ser eficiente. Começa por analisar as subsequências de x de comprimento 1, depois as de comprimento 2 e sucessivamente até n (isto é, até tomar a palavra completa). Para cada i e cada s tais que $1 \leq i \leq i + s \leq n$ e $s \geq 0$, denotemos por $N[i, i + s]$ o conjunto de variáveis em V que geram a subpalavra $x_i \ldots x_{i+s}$ de x, isto é, $N[i, i + s] = \{A \mid A \in V,\ A \Rightarrow_{\mathcal{G}}^\star x_i \ldots x_{i+s}\}$. O algoritmo CYK obtém $N[i, i + s]$ na iteração s.

Algoritmo CYK

- Para todo i tal que $1 \leq i \leq n$, inicializar $N[i,i] := \{A \mid A \in V,\ A \to x_i\}$, e para todo j tal que $j \neq i$ e $1 \leq j \leq n$, definir $N[i,j] := \emptyset$.
- Para cada s entre 1 e $n-1$ fazer

  Para cada i entre 1 e $n-s$, considerar $N[i,k]$ e $N[k+1, i+s]$, para todo k com $i \leq k \leq (i+s)-1$.

  Se existir $(A \to BC) \in P$ tal que $B \in N[i,k]$ e $C \in N[k+1, i+s]$, acrescentar A a $N[i, i+s]$.
- A palavra x está em $\mathcal{L}(\mathcal{G})$ se e só se $S \in N[1,n]$.

É usual representar a informação por uma matriz como a seguinte, estando $N[t, t+s]$ na coluna t e linha $\#s{+}1$.

$\#n$	$N[1,n]$					
$\#n{-}1$	$N[1,n{-}1]$	$N[2,n]$				
$\vdots$	$\vdots$	$\vdots$	$\vdots$			
$\#3$	$N[1,3]$	$N[2,4]$	$\cdots$	$N[n{-}2,n]$		
$\#2$	$N[1,2]$	$N[2,3]$	$\cdots$	$N[n{-}2,n{-}1]$	$N[n{-}1,n]$	
$\#1$	$N[1,1]$	$N[2,2]$	$\cdots$	$N[n{-}2,n{-}2]$	$N[n{-}1,n{-}1]$	$N[n,n]$
	x_1	x_2	$\cdots$	x_{n-2}	x_{n-1}	x_n

Importa recordar que a entrada $N[t, t+s]$ da tabela apresenta o conjunto das categorias possíveis para a subpalavra $x_t \cdots x_{t+s}$ de x.

$\#n$	$x_1 \cdots x_n$					
$\#n{-}1$	$x_1 \cdots x_{n-1}$	$x_2 \cdots x_n$				
$\vdots$	$\vdots$	$\vdots$	$\vdots$			
$\#3$	$x_1x_2x_3$	$x_2x_3x_4$	$\cdots$	$x_{n-2}x_{n-1}x_n$		
$\#2$	x_1x_2	x_2x_3	$\cdots$	$x_{n-2}x_{n-1}$	$x_{n-1}x_n$	
$\#1$	x_1	x_2	$\cdots$	x_{n-2}	x_{n-1}	x_n

Os elementos que intervêm no cálculo de $N[t, t+s]$ são os que estão abaixo desse elemento na coluna t e na diagonal descendente com origem nesse elemento. Assim, percorremos a coluna de baixo para cima e a diagonal de cima para baixo para formar os pares relevantes $N[t,t]N[t{+}1, t{+}s]$, $N[t,t{+}1]N[t{+}2, t{+}s]$, $N[t,t{+}2]N[t{+}3, t{+}s]$, $\ldots$, $N[t, t+s-1]N[t+s, t+s]$.

Exemplo 128 Consideremos a gramática $\mathcal{G}$, dada na forma normal de Chomsky, com conjunto de variáveis $V = \{E, T, F, E_1, E_2, T_1, T_2, T_3, M, S, X, Q, A, B\}$, símbolo inicial E, e seguintes produções:

$$
\begin{array}{lcl}
E & \to & TE_1 \mid TE_2 \mid FT_1 \mid FT_2 \mid AT_3 \mid \texttt{n} \\
T & \to & \texttt{n} \mid FT_1 \mid FT_2 \mid AT_3 \\
F & \to & AT_3 \mid \texttt{n} \\
E_1 & \to & ME \\
E_2 & \to & SE \\
T_1 & \to & XT \\
T_2 & \to & QT
\end{array}
\qquad
\begin{array}{lcl}
M & \to & + \\
S & \to & - \\
X & \to & \star \\
Q & \to & / \\
A & \to & (\\
B & \to &) \\
T_3 & \to & EB
\end{array}
$$

Vamos aplicar o algoritmo CYK para mostrar que $(\texttt{n} + \texttt{n}) * \texttt{n} \in \mathcal{L}(\mathcal{G})$, o que equivale a mostrar que $E \in N[1,7]$.

$s=1$	$s=2$	$s=3$	$s=4$
$N[1,1] = \{A\}$	$N[1,2] = \emptyset$	$N[1,3] = \emptyset$	
$N[2,2] = \{E,T,F\}$	$N[2,3] = \emptyset$	$N[2,4] = \{E\}$	$N[1,4] = \emptyset$
$N[3,3] = \{M\}$	$N[3,4] = \{E_1\}$	$N[3,5] = \emptyset$	$N[2,5] = \{T_3\}$
$N[4,4] = \{E,T,F\}$	$N[4,5] = \{T_3\}$	$N[4,6] = \emptyset$	$N[3,6] = \emptyset$
$N[5,5] = \{B\}$	$N[5,6] = \emptyset$	$N[5,7] = \emptyset$	$N[4,7] = \emptyset$
$N[6,6] = \{X\}$	$N[6,7] = \{T_1\}$		
$N[7,7] = \{E,T,F\}$			

$s=5$	$s=6$	$s=7$
$N[1,5] = \{F,T,E\}$	$N[1,6] = \emptyset$	
$N[2,6] = \emptyset$	$N[2,7] = \emptyset$	$N[1,7] = \{T,E\}$
$N[3,7] = \emptyset$		

#7	$\{T,E\}$						
#6	$\emptyset$	$\emptyset$					
#5	$\{F,T,E\}$	$\emptyset$	$\emptyset$				
#4	$\emptyset$	$\{T_3\}$	$\emptyset$	$\emptyset$			
#3	$\emptyset$	$\{E\}$	$\emptyset$	$\emptyset$	$\emptyset$		
#2	$\emptyset$	$\emptyset$	$\{E_1\}$	$\{T_3\}$	$\emptyset$	$\{T_1\}$	
#1	$\{A\}$	$\{E,T,F\}$	$\{M\}$	$\{E,T,F\}$	$\{B\}$	$\{X\}$	$\{E,T,F\}$
	(	n	$+$	n	)	$*$	n

Por exemplo, para obter $N[1,2]$, analisámos $N[1,1]N[2,2] = \{A\}\{E,T,F\} = \{AE, AT, AF\}$ e verificámos que nenhuma das três sequências, AE, AT, e AF, é lado direito de alguma regra. Portanto, $N[1,2] = \emptyset$.

Para $N[2,3]$, analisámos $N[2,2]N[2,3] = \{E,T,F\}\{M\} = \{EM, TM, FM\}$, concluindo que $N[2,3] = \emptyset$.

Para obter $N[3,4]$, analisámos $N[3,3]N[4,4] = \{M\}\{E,T,F\} = \{ME, MT, MF\}$ e verificámos que nenhuma apenas ME ocorre como lado direito de alguma regra (a regra $E_1 \to ME$). Portanto, $N[3,4] = \{E_1\}$.

No fim, para $N[1,7]$, tomámos $N[1,1]N[2,7] = \emptyset,\ N[1,2]N[3,7] = \emptyset,\ N[1,3]N[4,7] = \emptyset,\ N[1,4]N[5,7] = \emptyset,$ $N[1,5]N[6,7] = \{F,T,E\}\{T_1\} = \{FT_1, TT_1, ET_1\}$ e $N[1,6]N[7,7] = \emptyset$, e vimos que FT_1 é da categoria E ou T, e que TT_1 e ET_1 não correspondem a qualquer categoria (não ocorrem no lado direito de regras).

Exemplo 129 A linguagem $L = \{\mathtt{c}x\mathtt{c}y \mid x, y \in \{\mathtt{a},\mathtt{b}\}^\star \text{ e } 1 \leq |x| \leq 2|y|\}$ é gerada pela gramática

$$G_1 = (\{S, A, B, X\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, P_1, S)$$

com P_1 dado por:

$$\begin{array}{lcl} S & \rightarrow & \mathtt{c}XXAX \mid \mathtt{c}XBX \\ A & \rightarrow & XXAX \mid XBX \mid B \\ B & \rightarrow & BX \mid \mathtt{c} \\ X & \rightarrow & \mathtt{a} \mid \mathtt{b} \end{array}$$

As regras da gramática garantem que a palavra começa por $\mathtt{c}$ e que $|x| \geq 1$, e baseiam-se no seguinte. Como as palavras de L são da forma $\mathtt{c}x\mathtt{c}y$ com $1 \leq |x| \leq 2|y|$, qualquer símbolo de y pode dar origem a dois símbolos de x no máximo. Assim, dada uma palavra $\mathtt{c}x\mathtt{c}y$, para verificar se pertence a L, podemos começar por emparelhar dois símbolos de x com um de y, processando x da esquerda para a direita e y da direita para a esquerda, até esgotar x (caso $|x|$ par) ou só sobrar um símbolo de x (caso $|x|$ ímpar). Esse símbolo terá de ser ainda emparelhado com um símbolo de y. Finalmente, só restará uma subpalavra $\mathtt{c}y'$, com $y' \in \{\mathtt{a},\mathtt{b}\}^\star$. Assim, a primeira regra para A procede ao esgotamento dos símbolos de x, podendo sobrar ainda um símbolo (que será tratado pela segunda regra) ou nenhum (terceira regra). A variável B produzirá $\mathtt{c}y'$. Podemos ainda observar que a estrutura que fixámos para a decomposição de cada palavra garante que G_1 não é ambígua.

Para reduzir G_1 à forma normal de Chomsky, introduzimos uma nova variável C para não ter terminais no lado direito de regras que não são da forma $V \rightarrow t$, com $t \in \Sigma$. Removemos também a produção unitária $A \rightarrow B$, criando duas regras novas para A (que resultam da substituição de B em $A \rightarrow B$ pelo lado direito das produções de B).

$$\begin{array}{lcl} S & \rightarrow & CXXAX \mid CXBX \\ A & \rightarrow & XXAX \mid XBX \mid BX \mid \mathtt{c} \\ B & \rightarrow & BX \mid \mathtt{c} \\ C & \rightarrow & \mathtt{c} \\ X & \rightarrow & \mathtt{a} \mid \mathtt{b} \end{array}$$

Para obter uma gramática equivalente mas na forma normal de Chomsky, basta agora transformar todas as regras do tipo $V \rightarrow \gamma$ com $|\gamma| \geq 3$, usando variáveis auxiliares.

$$
\begin{array}{lcl}
S & \to & CR_1 \mid CW_1 \\
A & \to & XR_2 \mid XW_2 \mid BX \mid \texttt{c} \\
B & \to & BX \mid \texttt{c} \\
C & \to & \texttt{c} \\
X & \to & \texttt{a} \mid \texttt{b}
\end{array}
\qquad
\begin{array}{lcl}
R_1 & \to & XR_2 \\
R_2 & \to & XR_3 \\
R_3 & \to & AX \\
W_1 & \to & XW_2 \\
W_2 & \to & BX
\end{array}
$$

Para a gramática na forma normal de Chomsky, vamos aplicar o algoritmo CYK para averiguar se a palavra $\texttt{cabcbba} \in L$. A tabela encontra-se abaixo. Sendo $\texttt{cabcbba} = x_1x_2x_3x_4x_5x_6x_7$, a célula na linha i e coluna j contém o conjunto das possíveis categorias da subpalavra $x_j \ldots x_{j+i-1}$. Vemos que $\texttt{cabcbba} \in \mathcal{L}(G)$ pois S está no topo da tabela (i.e., na célula $(7,1)$, que caracteriza $x_1 \ldots x_7$).

#7	S						
#6	S	A, R_1, R_2, R_3					
#5	S	A, R_1, R_2, R_3	A, R_2, R_3, W_1				
#4	$\emptyset$	A, R_1	A, R_2, R_3, W_1	R_3, A, B, W_2			
#3	R_3, A, B, W_2	$\emptyset$	R_2, W_1, A	R_3, A, B, W_2	$\emptyset$		
#2	R_3, A, B, W_2	$\emptyset$	$\emptyset$	R_3, A, B, W_2	$\emptyset$	$\emptyset$	
#1	A, B, C	X	X	A, B, C	X	X	X
	c	a	b	c	b	b	a

A tabela foi construída da linha 1 para a linha 7. Por exemplo, para preencher a entrada $N[2,5]$, na linha 4 e coluna 2, considerámos $N[2,2]N[3,5] = \{X\}\{R_2, W_1, A\} = \{XR_2, XW_1, XA\}$, $N[2,3]N[4,5] = \emptyset\{R_3, A, B, W_2\} = \emptyset$, $N[2,4]N[5,5] = \emptyset\{X\} = \emptyset$. Só XR_2 é lado direito de regras (neste caso, de R_1 e A). Logo, $N[2,5] = \{R_1, A\}$.

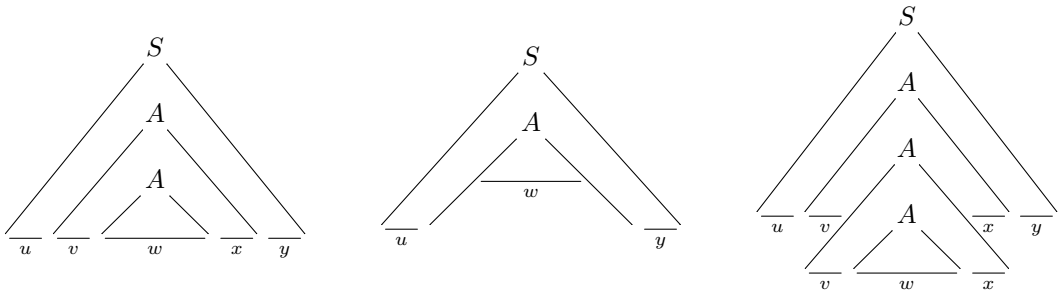
A tabela contém informação que pode ser usada para classificar subpalavras de cabcbba. Por exemplo, vemos que $\texttt{cbba} \notin L$, pois $x_4x_5x_6x_7 = \texttt{cbba}$ foi classificado como sendo das categorias R_3, A, B, ou W_2. Vemos também que $\texttt{cabcb} \in L$, pois $\texttt{cabcb} = x_1 \cdots x_5$ está classificado com categoria S (como se vê na linha 5 coluna 1). Vemos que $\texttt{bc} \notin L$ e não pode ser gerado por nenhuma variável da gramática (valor $\emptyset$ na entrada $(2,3)$).

6.5 Existência de linguagens que não são independentes de contexto

Existem linguagens que não são independentes de contexto. Por exemplo: $\{\texttt{0}^p \mid p \text{ primo}\}$ e $\{\texttt{0}^{n^2} \mid n \in \mathbb{N}\}$, de alfabeto $\{\texttt{0}\}$, $\{ww \mid w \in \Sigma^\star\}$, de alfabeto Σ, com $|\Sigma| \geq 2$, e $\{\texttt{0}^n\texttt{1}^n\texttt{2}^n \mid n \in \mathbb{N}\}$, de alfabeto $\{\texttt{0}, \texttt{1}, \texttt{2}\}$. O Lema da Repetição para LICs indica uma condição necessária para que uma linguagem seja independente de contexto.

Proposição 42 (Lema da Repetição para LICs) *Seja L uma linguagem independente de contexto. Então, existe uma constante $n \in \mathbb{N} \setminus \{0\}$, só dependente de L tal que qualquer que seja $z \in L$, se $|z| \geq n$, então podemos escrever $z = uvwxy$ de forma que $|vx| \geq 1$, $|vwx| \leq n$ e, para todo $i \in \mathbb{N}$, se tenha $uv^i wx^i y \in L$.*

Ideia da prova: Partindo de uma GIC $\mathcal{G}$ na forma normal de Chomsky (estendida) que gere L, basta tomar $n = 2^{|V|}$, para mostrar que qualquer sequência $z \in L$, com $|z| \geq n$, satisfaz as condições indicadas. De facto, estando $\mathcal{G}$ na forma de Chomsky, as árvores de derivação são árvores binárias, e é possível ver que se $|z| \geq n$, a árvore de derivação de z inclui uma subárvore A_1 com raíz A, para algum $A \in V$, que gera vwx com $|vwx| \leq n$ e $vx \neq \varepsilon$, e em que w é gerada por uma subárvore A_2 também com raíz A. Estas duas árvores podem ser usadas para mostrar $uv^i wx^i y \in L$, para todo $i \geq 0$. Se $i = 0$, transforma-se a árvore substituindo A_1 por A_2, o que dá origem a uma árvore de derivação para $uv^0 wx^0 y$. Se $i \geq 2$, substitui-se A_2 por A_1, sucessivamente, obtendo $uv^i wx^i y$ após i substituições.



A demonstração detalhada pode ser encontrada em [Hopcroft & Ullman]. □

Exemplo 130 A linguagem $L = \{ww \mid w \in \{0,1\}^\star\}$ não é independente de contexto.

Dado $n \geq 1$, tomamos $z = 0^{2n}1^{2n}0^{2n}1^{2n}$. A escolha simplifica o tipo de subpalavras vwx a analisar. Como $|vwx| \leq 1$, a subpalavra vwx abrange no máximo dois blocos da palavra, qualquer que seja vwx. Em qualquer caso, se se tomar $i = 0$, a palavra $uv^i wx^i y \notin L$. Por exemplo, se v for subpalavra do primeiro bloco de 0's, temos duas possibilidades:

$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v} w \underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y} \qquad \text{e} \qquad z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v} w \underbrace{0^{|x|}}_{x}\underbrace{y'1^{2n}0^{2n}1^{2n}}_{y}$$

No primeiro caso, a palavra $uv^0wx^0y = 0^{|u|}wy'0^{2n}1^{2n} = 0^{2n-|v|}1^{2n-|x|}0^{2n}1^{2n} \notin L$ porque, se dividirmos a palavra ao meio, a primeira metade termina em 0 e segunda termina em 1. No segundo caso, $uv^0wx^0y = 0^{2n-|v|-|x|}1^{2n}0^{2n}1^{2n} \notin L$, porque depois do meio da palavra há mais 0's do que na primeira metade.

Do mesmo modo, se v tiver 0's do primeiro bloco e algum 1 do segundo, então x só poderá ter 1's do segundo bloco, sendo $z = \underbrace{0^{|u|}}_{u}\underbrace{0^k1^{|v|-k}}_{v} w \underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y}$, para algum k, e $|u| > n$. Se cortarmos v e x, a palavra uv^0wx^0y tem mais 0's no segundo bloco de 0's do que no primeiro.

Os restantes casos podem ser analisados de forma análoga.

Exemplo 131 A linguagem $L = \{\mathtt{a}^k\mathtt{b}^k\mathtt{c}^k \mid k \in \mathbb{N}\}$ não é independente de contexto.

Para o mostrar, vamos verificar que L não satisfaz a condição do lema da repetição para LICs.

Dado $n \geq 1$, escolhemos $z = \mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$. Esta escolha simplifica o tipo de subpalavras que vamos ter que analisar, porque sendo $|vwx| \leq n$, garantimos que não temos simultaneamente a's, b's e c's em vwx. Claramente, $z \in L$ e $|z| = 3n \geq n$. Não existem $u, v, w, x, y \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^\star$ tais que

$$\left|\begin{array}{l} \mathtt{a}^n\mathtt{b}^n\mathtt{c}^n = uvwxy, \\ vx \neq \varepsilon, \\ |vwx| \leq n, \\ \text{e para todo } i \in \mathbb{N}, \text{ se tenha } uv^iwx^iy \in L. \end{array}\right.$$

Como já observámos, para se ter $|vwx| \leq n$, em vwx não podem existir simultanemente a's, b's e c's. Qualquer que seja a decomposição de z como $uvwxy$ com $|vx| \neq 0$ e $|vwx| \leq n$, para $i = 0$ (*cortar* v e x), $uv^0wx^0y \notin L$ porque não tem igual número de a's, b's e c's.

Proposição 43 *A classe de LICs não é fechada para a complementação nem para a intersecção.*

Prova: Tomamos $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ e consideramos as linguagens independentes de contexto L_1 e L_2 dadas por $L_1 = \{\mathtt{a}^k\mathtt{b}^k\mathtt{c}^m \mid k, m \in \mathbb{N}\} = \mathcal{L}(\mathcal{G}_1)$ e $L_2 = \{\mathtt{a}^m\mathtt{b}^k\mathtt{c}^k \mid k, m \in \mathbb{N}\} = \mathcal{L}(\mathcal{G}_2)$, para

$$\mathcal{G}_1 = (\{S, A, C\}, \Sigma, \{S \to AC, A \to \mathtt{a}A\mathtt{b}, A \to \varepsilon, C \to \varepsilon, C \to \mathtt{c}C\}, S)$$

$$\mathcal{G}_2 = (\{S, A, C\}, \Sigma, \{S \to CA, A \to \mathtt{b}A\mathtt{c}, A \to \varepsilon, C \to \varepsilon, C \to \mathtt{a}C\}, S).$$

Como $L_1 \cap L_2 = \{\mathtt{a}^k\mathtt{b}^k\mathtt{c}^k \mid k \in \mathbb{N}\}$ e já provámos que esta linguagem não é independente de contexto, concluimos que existem linguagens independentes de contexto cuja intersecção não é independente de contexto.

Como consequência, a classe das LICs não é fechada para a complementação. Como, $\overline{\overline{A} \cup \overline{B}} = A \cap B$ e a classe de LICs é fechada para a união, se fosse fechada para a complementação, então seria fechada para a intersecção. □

Observação. Como qualquer linguagem regular é independente de contexto, se mostrarmos que uma dada linguagem não satisfaz a condição do Lema da Repetição para LICs, podemos imediatamente concluir que não é regular.

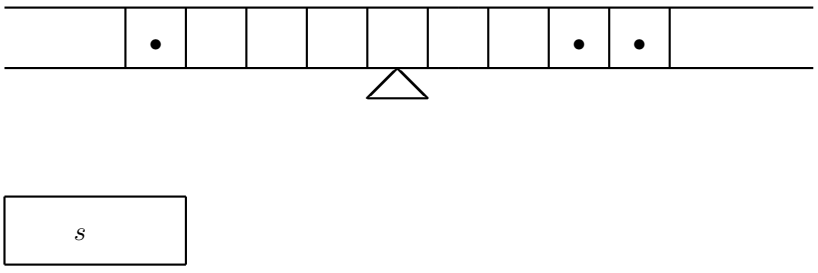
Capítulo 7

Máquinas de Turing

A **máquina de Turing** (1936) foi proposta como um modelo matemático para "métodos algorítmicos". Intuitivamente, um método algorítmico é *uma qualquer sequência finita de instruções que possa ser executada mecanicamente*. Esta noção não tem precisão matemática, pelo que não se conseguiu demonstrar que seja equivalente ao modelo (formal) de máquina de Turing. As máquinas de Turing são tão gerais que parecem poder simular qualquer computação possível.

Outros modelos de computação foram propostos com o mesmo objectivo, mas nenhum desses modelos é mais geral do que a máquina de Turing (alguns são equivalentes). Por isso, embora não tenha sido demonstrada, aceita-se a **conjetura** (ou seja, **hipótese**) **de Church–Turing** que diz que existe um método algorítmico para resolver um dado problema se e só se existir uma máquina de Turing para o resolver.

Existem vários modelos de máquina de Turing. Aquele que vamos descrever pode ser visto como o modelo duma máquina que dispõe duma fita dividida em células, a qual é infinita para a esquerda e para a direita.



A máquina tem uma cabeça de leitura/escrita, que pode deslocar ao longo da fita. Em cada transição pode efetuar um deslocamento para a esquerda ou para a direita duma só posição, ou seja passar à célula imediatamente à esquerda (respectivamente, à direita). Cada célula pode conter um símbolo. A máquina pode ler e escrever símbolos na fita.

Como nos modelos de computação que vimos anteriormente, a máquina pode estar num dado estado, sendo o conjunto de estados **finito**.

Como a fita é infinita, usa-se um símbolo particular, que se diz **símbolo branco**, para delimitar a parte da fita que contém a informação relevante. Na figura anterior, esse símbolo é •. Considera-se que todas as posições vazias têm •. Se num deslocamento para a direita (respectivamente, esquerda) encontrar o símbolo branco, é de "esperar" que a fita esteja vazia a partir daí.

Máquina de Turing como computador

Uma máquina de Turing pode ser vista como uma calculadora de algumas funções parciais de inteiros em inteiros, as quais se dizem *funções parcialmente recursivas*. Se se mostrar que uma dada máquina de Turing que calcula uma tal função pára para todos os dados possíveis, então a função diz-se *recursiva*.

Máquina de Turing como reconhecedor

Dizemos que **uma máquina de Turing aceita uma palavra** x dum dado alfabeto Σ se e só se dada a palavra x na fita, estando a cabeça de leitura posicionada no símbolo de x mais à esquerda, e estando a máquina no **estado inicial**, esta efetua transições de acordo com a sua descrição e **pára num estado final** (ou seja, quando não pode efetuar mais transições está em estado final). Se a máquina for não-determinística, dizemos que x é aceite, se existir alguma sequência de transições que leve a máquina a parar em estado final.

Definição 18 *Uma máquina de Turing $\mathcal{M}$ é um sistema $\mathcal{M} = (S, \Sigma, \Gamma, \delta, s_0, b, F)$, onde S é o* **conjunto de estados**, *S é finito, Γ é o* **alfabeto da fita**, *$\Sigma \subset \Gamma$ é o* **alfabeto de entrada**, *$s_0 \in S$ é o* **estado inicial**, *$b \in \Gamma$ o* **símbolo branco**, *$F \subseteq S$ o conjunto de* **estados finais**, *e δ a* **relação de transição** *é uma relação binária de $S \times \Gamma$ em $S \times \Gamma \times \{e, d\}$. Convencionamos que e indica* **movimento para esquerda** *e d indica* **movimento para direita**.

A linguagem reconhecida (ou aceite) **pela máquina de Turing** $\mathcal{M}$ é o conjunto das palavras aceites por $\mathcal{M}$, denotando-se por $\mathcal{L}(\mathcal{M})$.

Pode acontecer, que uma dada máquina de Turing $\mathcal{M}$ não pare (entra em computação infinita) quando uma certa sequência $x \in \Sigma^\star$ lhe for dada. Uma linguagem $L \subseteq \Sigma^\star$ diz-se **decidível** (ou **recursiva**) se e só se existir uma máquina de Turing $\mathcal{M}$ tal que $L = \mathcal{L}(\mathcal{M})$ e $\mathcal{M}$ pára para todos os dados (isto é, para todo $x \in \Sigma^\star$, a máquina $\mathcal{M}$ pára quando x é dado inicialmente na fita, e o estado em que parou é final se e só se $x \in L$).

Uma linguagem $L \subseteq \Sigma^\star$ diz-se **semi-decidível** (ou **recursivamente enumerável**) se e só se existir uma máquina de Turing $\mathcal{M}$ tal que $L = \mathcal{L}(\mathcal{M})$. Dado $x \in \Sigma^\star \setminus L$ a máquina pode não parar (mas pára para todo $x \in L$).

Exemplo 132 Vamos justificar que $\{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^n \mid n \in \mathbb{N}\}$ é decidível, ou seja, definir uma máquina de Turing que dado $x \in \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}^\star$ indica se x é, ou não é, da linguagem $\{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^n \mid n \in \mathbb{N}\}$.

A máquina preservará a palavra dada na fita se esta for da linguagem. Caso contrário, pode destruí-la. Vamos usar nomes para os estados que nos facilitem a compreensão da máquina. O estado inicial é $inicio$, o símbolo branco é •, e representamos cada transição por um quinteto (s, a, s', a', m), onde s é o estado em que a máquina está, a o símbolo que está na célula atual, s' o estado a que passa, a' o símbolo que passa a estar na célula e $m \in \{e, d\}$ é o sentido do movimento que faz. O estado final será $aceita$.

A ideia geral do método é: por cada `0` que encontra, a máquina vai para a direita "cortar" um `1` e depois "cortar" um `2`. A seguir, volta para a esquerda à procura do último `0` que viu, o qual deixou marcado com `Z`. Quando o encontrar, segue para a posição imediatamente à direita e, se o símbolo que lá está ainda for `0`, então repete a procura de `1` e de `2`. Se não for `0`, segue para a direita para verificar se todos os símbolos até ao • estão "cortados". De facto, "cortado" quer dizer "marcado", sendo os `1`'s marcados com `U`'s e os `2`'s com `B`'s, para que seja possível no fim repor a sequência dada.

$(inicio,$ •, $aceita,$ •, $e)$	*Se tem* • *no início, a sequência é* ε
$(inicio,$ `0`, $proc1,$ `Z`, $d)$	*Marca o* `0` *com* `Z` *e vai procurar um* `1`.
$(proc1,$ `0`, $proc1,$ `0`, $d)$	*Ignora os* `0`*'s quando procura* `1`*'s.*
$(proc1,$ `1`, $proc2,$ `U`, $d)$	*Quando encontra* `1`*, marca-o e vai procurar* `2`*.*
$(proc2,$ `1`, $proc2,$ `1`, $d)$	*Ignora os* `1`*'s quando procura* `2`*'s.*
$(proc2,$ `2`, $procZ,$ `B`, $e)$	*Quando encontra* `2`*, marca-o e vai procurar* `0`*'s para a esquerda.*
	O último `0` *que viu está marcado com um* `Z`*, que vai procurar.*
$(procZ,$ `B`, $procZ,$ `B`, $e)$	*Passa os* `B`*'s quando procura o* `Z`*.*
$(procZ,$ `U`, $procZ,$ `U`, $e)$	*Passa os* `U`*'s quando procura o* `Z`*.*
$(procZ,$ `1`, $procZ,$ `1`, $e)$	*Passa os* `1`*'s quando procura o* `Z`*.*
$(procZ,$ `0`, $procZ,$ `0`, $e)$	*Passa os* `0`*'s quando procura o* `Z`*.*
$(procZ,$ `Z`, $proc0,$ `Z`, $d)$	*Encontra* `Z`*. Ainda há* `0`*'s?*
$(proc0,$ `0`, $proc1,$ `Z`, $d)$	*Se encontra* `0`*, repete o processo*
$(proc0,$ `U`, $verif,$ `U`, $d)$	*Não há mais* `0`*'s! Foram cortados todos os símbolos?*
$(verif,$ `U`, $verif,$ `U`, $d)$	*Só pode haver* `U`*'s ou* `B`*'s até* •
$(verif,$ `B`, $verif,$ `B`, $d)$	
$(verif,$ •, $repor,$ •, $e)$	A palavra é da linguagem; vai para a esquerda repô-la.

$(repor,\ \texttt{B},\ repor,\ \texttt{2},\ e)$

$(repor,\ \texttt{U},\ repor,\ \texttt{1},\ e)$

$(repor,\ \texttt{Z},\ repor,\ \texttt{0},\ e)$

$(repor,\ \bullet,\ aceita,\ \bullet,\ d)$ Pára em $aceita$ na posição inicial.

$(proc1,\ \texttt{U},\ proc1,\ \texttt{U},\ d)$ À procura de 1's, tem que poder ignorar os já cortados.

$(proc2,\ \texttt{B},\ proc2,\ \texttt{B},\ d)$ À procura de 2's, tem que poder ignorar os já cortados.

As duas últimas transições são necessárias para evitar que a máquina encrave (isto é, páre) erradamente antes de chegar a estado final, por não ter possibilidade de passar por cima das marcações que deixámos na fita.

Em conclusão, $S = \{inicio, aceita, proc1, proc2, procZ, proc0, repor, verif\}$, $F = \{aceita\}$, estado inicial $inicio$, $\Gamma = \{\bullet, \texttt{U}, \texttt{B}, \texttt{Z}\} \cup \{\texttt{0}, \texttt{1}, \texttt{2}\}$, as transições são as dadas acima e $\bullet$ é o símbolo branco.

Vamos ilustrar o funcionamento da máquina através da análise das transições para `001122`. Para representar cada configuração, representamos a parte relevante da fita, e *inserimos o estado imediatamente à esquerda* do símbolo que está na célula atual.

1 : $\cdots \bullet \bullet$ `inicio` `001122` $\bullet \bullet \cdots$

2 : $\cdots \bullet \bullet$`Z` `proc1` `01122` $\bullet \bullet \cdots$

3 : $\cdots \bullet \bullet$`Z0` `proc1` `1122` $\bullet \bullet \cdots$

4 : $\cdots \bullet \bullet$`Z0U` `proc2` `122` $\bullet \bullet \cdots$

5 : $\cdots \bullet \bullet$`Z0U1` `proc2` `22` $\bullet \bullet \cdots$

6 : $\cdots \bullet \bullet$`Z0U` `procZ` `1B2` $\bullet \bullet \cdots$

7 : $\cdots \bullet \bullet$`Z0` `procZ` `U1B2` $\bullet \bullet \cdots$

8 : $\cdots \bullet \bullet$`Z` `procZ` `0U1B2` $\bullet \bullet \cdots$

9 : $\cdots \bullet \bullet$ `procZ` `Z0U1B2` $\bullet \bullet \cdots$

10 : $\cdots \bullet \bullet$`Z` `proc0` `0U1B2` $\bullet \bullet \cdots$

11 : $\cdots \bullet \bullet$`ZZ` `proc1` `U1B2` $\bullet \bullet \cdots$

12 : $\cdots \bullet \bullet$`ZZU` `proc1` `1B2` $\bullet \bullet \cdots$

13 : $\cdots \bullet \bullet$`ZZUU` `proc2` `B2` $\bullet \bullet \cdots$

14 : $\cdots \bullet \bullet$`ZZUUB` `proc2` `2` $\bullet \bullet \cdots$

15 : $\cdots \bullet \bullet$`ZZUU` `procZ` `BB` $\bullet \bullet \cdots$

16 : $\cdots \bullet \bullet$`ZZU` `procZ` `UBB` $\bullet \bullet \cdots$

17 : $\cdots \bullet \bullet$`ZZ` `procZ` `UUBB` $\bullet \bullet \cdots$

18 : $\cdots \bullet \bullet$`Z` `procZ` `ZUUBB` $\bullet \bullet \cdots$

19 : $\cdots \bullet \bullet$`ZZ` `proc0` `UUBB` $\bullet \bullet \cdots$

20 : $\cdots \bullet \bullet$`ZZU` `verif` `UBB` $\bullet \bullet \cdots$

21 : $\cdots \bullet \bullet$`ZZUU` `verif` `BB` $\bullet \bullet \cdots$

22 : $\cdots \bullet \bullet$`ZZUUB` `verif` `B` $\bullet \bullet \cdots$

23 : $\cdots \bullet \bullet$`ZZUUBB` `verif` $\bullet \bullet \cdots$

23 : $\cdots \bullet \bullet$`ZZUUB` `repor` `B` $\bullet \bullet \cdots$

24 : $\cdots \bullet \bullet$`ZZUU` `repor` `B2` $\bullet \bullet \cdots$

25 : $\cdots \bullet \bullet$`ZZU` `repor` `U22` $\bullet \bullet \cdots$

26 : $\cdots \bullet \bullet$`ZZ` `repor` `U122` $\bullet \bullet \cdots$

27 : $\cdots \bullet \bullet$`Z` `repor` `Z1122` $\bullet \bullet \cdots$

28 : $\cdots \bullet \bullet$ `repor` `Z01122` $\bullet \bullet \cdots$

29 : $\cdots \bullet$ `repor` $\bullet$ `001122` $\bullet \bullet \cdots$

30 : $\cdots \bullet \bullet$ `aceita` `001122` $\bullet \bullet \cdots$

Se estivessemos a analisar a palavra `0011122`, quando a máquina entrasse em `verif`, a configuração seria

$$\cdots \bullet \bullet \texttt{ZZU}\boxed{\texttt{verif}}\texttt{U1BB}\bullet\bullet\cdots$$

e a seguinte

$$\cdots \bullet \bullet \texttt{ZZUU}\boxed{\texttt{verif}}\texttt{1BB}\bullet\bullet\cdots$$

o que a levaria a parar, pois não tem transição em `verif` se o símbolo na célula for `1`. Como esse estado não é final, não aceitava a palavra.

Se estivessemos a analisar a palavra `0011022`, a máquina pára em `proc2`, porque não pode passar por cima de `0`'s quando procura um `2`. A configuração final seria

$$\cdots \bullet \bullet \texttt{Z0U1}\boxed{\texttt{proc2}}\texttt{022}\bullet\bullet\cdots$$

Como esse estado não é final, não aceitava a palavra.

Preservar sempre a sequência dada na fita. Vamos modificar a máquina de modo a manter, em todos os casos, a sequência de $\{\texttt{0},\texttt{1},\texttt{2}\}^\star$ dada na fita. Para isso, vamos incluir transições para que, quando a máquina "descobrir" que a palavra não é da linguagem, volte a repor os símbolos que substituiu. Em $limpar$ a "limpeza" vai ser feita da direita para a esquerda.

$(proc1,\ \texttt{2},\ limpar,\ \texttt{2},\ d)$
$(proc2,\ \texttt{0},\ limpar,\ \texttt{0},\ d)$
$(verif,\ \texttt{1},\ limpar,\ \texttt{1},\ d)$
$(verif,\ \texttt{2},\ limpar,\ \texttt{2},\ d)$
$(verif,\ \texttt{0},\ limpar,\ \texttt{0},\ d)$ Este `0` está depois de um `U`

$(limpar,\ \texttt{0},\ limpar,\ \texttt{0},\ d)$
$(limpar,\ \texttt{1},\ limpar,\ \texttt{1},\ d)$
$(limpar,\ \texttt{2},\ limpar,\ \texttt{2},\ d)$
$(limpar,\ \texttt{B},\ limpar,\ \texttt{B},\ d)$
$(limpar,\ \texttt{U},\ limpar,\ \texttt{U},\ d)$
$(limpar,\ \bullet,\ reporn\ \bullet,\ e)$

$(repor\_n,\ \texttt{1},\ repor\_n,\ \texttt{1},\ e)$
$(repor\_n,\ \texttt{2},\ repor\_n,\ \texttt{2},\ e)$
$(repor\_n,\ \texttt{B},\ repor\_n,\ \texttt{2},\ e)$
$(repor\_n,\ \texttt{U},\ repor\_n,\ \texttt{1},\ e)$
$(repor\_n,\ \texttt{Z},\ repor\_n,\ \texttt{0},\ e)$
$(repor\_n,\ \texttt{0},\ repor\_n,\ \texttt{0},\ e)$
$(repor\_n,\ \bullet,\ naoaceita,\ \bullet,\ d)$

As palavras que começam por `0` e não são da linguagem, levam a máquina ao estado $naoaceita$ (parando na célula em que estava inicialmente). As palavras que começam por `1` ou `2` fazem a máquina parar em $inicio$ (sem efetuar qualquer transição).

Vamos agora dar exemplo duma máquina de Turing que calcula o valor duma função.

Exemplo 133 Vamos definir uma máquina de Turing que, dada uma sequência da forma $x\mathtt{0}y$, com $x, y \in \{\mathtt{1}\}\{\mathtt{1}\}^\star$, determina uma sequência $q\mathtt{0}r$ em que $q, r \in \{\mathtt{1}\}^\star$ e $|q|$ e $|r|$ são o quociente e o resto da divisão inteira de $|x|$ por $|y|$, respectivamente.

Do mesmo modo que o produto de dois inteiros pode ser calculado por somas sucessivas, a divisão pode ser calculada por subtracções sucessivas.

Assim, por exemplo, para calcularmos o quociente da divisão de 10 por 4, dada a sequência `111111111101111`, devemos ver quantas vezes se pode cortar a sequência `1111` na sequência `1111111111`. Na máquina que vamos definir, marcamos com um `X` o `1` de x que é cortado no início dum novo processo de corte de y em x (ou no que resta de x depois de cortes sucessivos). Quando y já não couber no que ainda não foi cortado em x, então o número de `X`'s à esquerda de `0` é o quociente da divisão inteira se o resto for zero. Caso contrário, excede em uma unidade esse quociente

Vamos assumir, sem verificar, que a sequência dada na fita é da forma $x\mathtt{0}y$, com $x, y \in \{\mathtt{1}\}\{\mathtt{1}\}^\star$. Supomos que o estado inicial é $inicio$ e o estado final $stop$.

$(inicio, \mathtt{1}, procy, \mathtt{X}, d)$	`X` *indica o primeiro* `1` *que corta em* x *por cada bloco* $\|y\|$
	O número de `X`*'s da' o quociente se o resto for zero.*
$(procy, \mathtt{1}, procy, \mathtt{1}, d)$	*Cortar o primeiro de* y
$(procy, \mathtt{0}, primy, \mathtt{0}, d)$	
$(primy, \mathtt{1}, maisy, \mathtt{Y}, d)$	*Corta o primeiro e vai cortar o segundo também.*
	`B` *marca último* `1` *de* y *que cortou*
$(maisy, \mathtt{1}, maisx, \mathtt{B}, e)$	*Em* `maisx`*, procurar* `1` *em* x *para emparelhar;* $\|y\| > 1$
$(maisy, \bullet, primx, \bullet, e)$	*Em* `maisy`*, está à procura do próximo* `1` *em* y

No estado `primx`, verifica se ainda há mais algum `1` em x. Para isso, vai procurar em x o `1` ainda não cortado que esteja o mais à esquerda possível. Quando vai para a esquerda para fazer essa tarefa, vai passar em cima de y,

aproveitando para repor y, substituindo cada `Y` por `1`.

$(primx, \texttt{Y}, primx, \texttt{1}, e)$	
$(primx, \texttt{0}, primx, \texttt{0}, e)$	
$(primx, \texttt{1}, primx, \texttt{1}, e)$	
$(primx, \texttt{A}, proxX, \texttt{A}, d)$	*Encontrou o último* `1` *que foi cortado em* x.
$(primx, \texttt{X}, proxX, \texttt{X}, d)$	*Se* $y = \texttt{1}$*, em cada corte dum bloco* y*, só um* `1` *é cortado em* x.
$(proxX, \texttt{1}, procy, \texttt{X}, d)$	*Tentar encaixar outro bloco* y *em* x.
$(proxX, \texttt{0}, resto0, \texttt{0}, d)$	*O resto é zero!*

No caso de y ter vários 1's, cada um deles vai ser cortado em x. No estado $maisx$ está a tentar emparelhar um dos 1's de y com algum de x.

$(maisx, \texttt{0}, maisx, \texttt{0}, e)$	
$(maisx, \texttt{1}, maisx, \texttt{1}, e)$	
$(maisx, \texttt{Y}, maisx, \texttt{Y}, e)$	
$(maisx, \texttt{X}, cortax, \texttt{X}, d)$	*O último* `1` *cortado em* x.
$(maisx, \texttt{A}, cortax, \texttt{A}, d)$	*O último* `1` *cortado em* x.
$(cortax, \texttt{1}, outrosy, \texttt{A}, d)$	*Emparelhou um* `1` *de* y *com um* `1` *em* x*. Mais* `1`*'s em* y*?*
$(cortax, \texttt{0}, sobras, \texttt{1}, e)$	*O resto é pelo menos um.*

Quando x termina sem ter conseguido emparelhar o último `1` que cortou em y, então o resto não é zero. Neste caso, $|y| \geq 2$ e o resto da divisão é dado pelo número de símbolos à esquerda de `0` até ao último `X` escrito (ou seja, aquele que está mais à direita). A transição $(cortax, \texttt{0}, sobras, \texttt{1}, e)$, substitui o `0` por um `1`. Para separar o quociente do resto, vamos colocar no `X` referido um `0`. Todos os `A`'s até esse `X` são trocados por `1`'s. Desta forma, ficamos com o resto a seguir ao `0`. Para ficar só o resto, vamos posteriormente limpar toda a parte do y (colocando •'s).

$(sobras, \texttt{A}, sobras, \texttt{1}, e)$	
$(sobras, \texttt{X}, quoc, \texttt{0}, e)$	*Em* `quoc` *vai recolher os outros* `X`*'s para formar o quociente.*

Antes de analisarmos a parte final, vamos ver as transições para estado `outrosy`, quando a meio do corte do bloco y, e depois de emparelhar um dado `1` de y com um `1` de x, a máquina tenta ver se sobram `1`'s em y.

$(outrosy, \texttt{1}, outrosy, \texttt{1}, d)$	
$(outrosy, \texttt{0}, outrosy, \texttt{0}, d)$	
$(outrosy, \texttt{Y}, outrosy, \texttt{Y}, d)$	
$(outrosy, \texttt{B}, maisy, \texttt{Y}, d)$	Encontra o último `1` que cortou em y.

Quando y já não cabe em x, sabemos se o resto da divisão é zero ou não. Se for zero, o autómato entra no estado `resto0`. Caso contrário, entra no estado `sobras`.

O autómato entra no estado `resto0` ao efetuar a transição $(proxX, \texttt{0}, resto0, \texttt{0}, d)$, ficando à direita de `0`. Nessa fase, y só tem `1`'s. Para tratar o quociente de forma análoga quando o resto é zero e quando não é, fazemos

$$(resto0, \texttt{1}, faltaq, \texttt{Y}, e)$$
$$(faltaq, \texttt{0}, quoc, \texttt{0}, e)$$

para substituir o primeiro `1` de y por um `Y`. Note que quando o resto não é zero, a máquina descobre esse facto quando está a tentar cortar mais uma vez y em x, e já cortou pelo menos o primeiro e segundo `1` de y, sendo o primeiro sempre re-escrito em `Y`.

Vamos definir a recolha dos `X`'s para formar o quociente. Relembre que cada `X` representa o ínicio de mais uma tentativa de corte de y em x. No estado `quoc`, o resto já foi processado e por isso, o número de X's na fita, deve ser igual ao núemeero de `1`'s que a represetação do quociente vai ter. Note que os `A`'s representam `1`'s que foram cortados em x mas que não têm agora qualquer interesse.

$(quoc, \texttt{A}, quoc, \texttt{A}, e)$	*Continua à procura de um* `X`.
$(quoc, \texttt{X}, incrq, \texttt{A}, d)$	*"Apaga"* `X` *e vai juntar um* `1` *ao quociente.*
$(quoc, \bullet, limpa, \bullet, d)$	*Já não há mais* `X`*'s. Vai limpar os símbolos "estranhos".*
$(incrq, \texttt{A}, incrq, \texttt{A}, d)$	
$(incrq, \texttt{1}, mais1q, \texttt{1}, e)$	
$(incrq, \texttt{0}, mais1q, \texttt{0}, e)$	*Só é usada na primeira vez.*
$(mais1q, \texttt{A}, quoc, \texttt{1}, e)$	

Quando detecta que já não há mais `X`'s na fita, o cursor está no símbolo branco imediatamente à esquerda da posição em que o cursor estava inicialmente. Na fita tem uma sequência da forma $\texttt{A}^n\texttt{1}^p\texttt{01}^s\texttt{Y}\alpha$, em que α pode ter `1`'s, `Y`'s e um `B`. Tem que deixar na fita apenas $\texttt{1}^p\texttt{01}^s$ e por isso vai apagar os `A`'s, o `Y` e α com $\bullet$'s.

$(limpa, \texttt{A}, limpa, \bullet, d)$	
$(limpa, \texttt{1}, limpa, \texttt{1}, d)$	
$(limpa, \texttt{0}, limpa, \texttt{0}, d)$	
$(limpa, \texttt{Y}, apaga, \bullet, d)$	
$(apaga, \texttt{Y}, apaga, \bullet, d)$	
$(apaga, \texttt{B}, apaga, \bullet, d)$	
$(apaga, \texttt{1}, apaga, \bullet, d)$	
$(apaga, \bullet, paraI, \bullet, e)$	Vai colocar o cursor na primeira posição do resultado.

Para colocar o cursor no início do resultado, a máquina tem ainda as transições seguintes

$$(paraI, \bullet, paraI, \bullet, e) \qquad (paraI, \mathtt{1}, paraI, \mathtt{1}, e) \qquad (paraI, \mathtt{0}, cursor, \mathtt{0}, e)$$
$$(cursor, \mathtt{1}, cursor, \mathtt{1}, e) \qquad (cursor, \bullet, stop, \bullet, e)$$

Vamos tentar ilustrar o comportamento da máquina quando é dada na fita a palavra `111111111101111`.

1 : $\cdots \bullet\bullet$ `inicio` `111111111101111` $\bullet\bullet\cdots$

2 : $\cdots \bullet\bullet$ `X` `procy` `11111111101111` $\bullet\bullet\cdots$

3 : $\cdots \bullet\bullet$ `X1` `procy` `1111111101111` $\bullet\bullet\cdots$

4 : $\cdots \bullet\bullet$ `X11` `procy` `111111101111` $\bullet\bullet\cdots$

$\vdots$

$\cdots \bullet\bullet$ `X111111111` `procy` `01111` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `X1111111110` `primy` `1111` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `X1111111110Y` `maisy` `111` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `X1111111110` `maisx` `YB11` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `X111111111` `maisx` `0YB11` $\bullet\bullet\cdots$

$\vdots$

$\cdots \bullet\bullet$ `X` `maisx` `1111111110YB11` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `maisx` `X1111111110YB11` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `X` `cortax` `1111111110YB11` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `XA` `outrosy` `111111110YB11` $\bullet\bullet\cdots$

$\vdots$

$\cdots \bullet\bullet$ `XA111111110Y` `outrosy` `B11` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `XA111111110YY` `maisy` `11` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `XA111111110Y` `maisx` `YB1` $\bullet\bullet\cdots$

$\vdots$

$\cdots \bullet\bullet$ `X` `maisx` `A111111110YYB1` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `XA` `cortax` `11111110YYB1` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `XAA` `outrosy` `1111110YYB1` $\bullet\bullet\cdots$

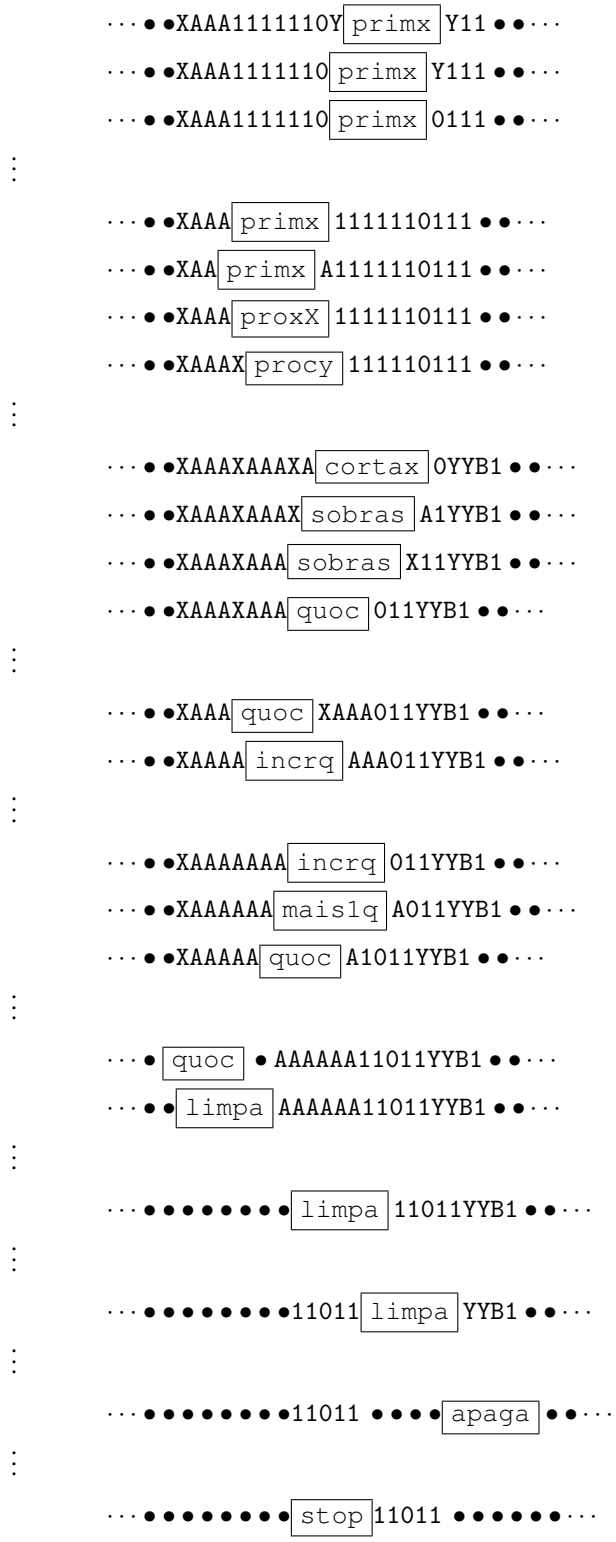
$\vdots$

$\cdots \bullet\bullet$ `XAAA1111110YYY` `outrosy` `B` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `XAAA1111110YYYY` `maisy` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `XAAA1111110YYY` `primx` `Y` $\bullet\bullet\cdots$

$\cdots \bullet\bullet$ `XAAA1111110YY` `primx` `Y1` $\bullet\bullet\cdots$



Proposição 44 *A classe das linguagens independentes de contexto está propriamente contida na classe das linguagens decidíveis. A classe das linguagens decidíveis está propriamente contida na classe das linguagens recursivamente enumeráveis.*

Qualquer linguagem decidível é, por definição, recursivamente enumerável. A demonstração de que existem linguagens não decídiveis mas que são recursivamente enumeráveis está fora do âmbito deste texto.

Podemos notar que um autómato de pilha pode ser simulado por uma máquina de Turing. Se partirmos de um autómato de pilha que corresponda a uma gramática na forma de Greibach, então, em cada transição, o autómato consome símbolos da palavra, o que permite garantir que a máquina de Turing termina num número finito de passos. Em alternativa, observemos que, quando $\varepsilon \notin L$, o algoritmo CYK constitui um algoritmo de decisão para $x \in L$, se se tiver uma gramática independente de contexto na forma de Chomsky que gere L (e pode ser adaptado ao caso geral).

Sabemos que as linguagens $\{\mathtt{0}^p \mid p \text{ primo}\}$ e $\{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^n \mid n \in \mathbb{N}\}$ não são independentes de contexto. Contudo, são linguagens decidíveis. Portanto, a inclusão da classe de LICs na classe de linguagens decidíveis é estrita.

Como referimos acima, dado um problema, é importante saber se existe um algoritmo para o resolver.
Foi demonstrada a não existência de algoritmos para resolver alguns problemas. Por exemplo:

- *O problema de paragem:* Não existe uma máquina de Turing que receba o "código" de outra qualquer máquina de Turing e diga se esta pára ou não. Do mesmo modo, não existe um programa de computador, que analise o código dum outro programa de computador e diga se este pára ou não.

- Existe um algoritmo para determinar o autómato finito determinístico mínimo que é equivalente a um dado autómato finito. Assim, podemos dizer que existe um algoritmo para verificar se dois autómatos finitos quaisquer são equivalentes.

  Mas, não existe um algoritmo que, dadas duas quaisquer gramáticas $\mathcal{G}$ e $\mathcal{G}'$ independentes de contexto, determine se as gramáticas são ou não são equivalentes (ou seja, se $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$). Não existe um algoritmo que, dados dois autómatos de pilha verifique se reconhecem ou não a mesma linguagem.

- Dada uma gramática independente de contexto $\mathcal{G}$, determinar se $\mathcal{L}(\mathcal{G}) = \Sigma^\star$ é um problema indecidível.

Bibliografia

[Lewis & Papadimitriou] Lewis, Papadimitriou, *Elements of the theory of Computation*, 2nd edition, Prentice Hall, 1998.

[Hopcroft & Ullman] Hopcroft, Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.