Modelos de Computação

Ana Paula Tomás e Nelma Moreira
Departamento de Ciência de Computadores
Faculdade de Ciências, Universidade do Porto
email: apt@ncc.up.pt,nam@ncc.up.pt

2005

Conteúdo

Capítulo 1

Preliminares

1.1 Notação para conjuntos

Nesta secção vamos essencialmente rever alguma da notação que usaremos para conjuntos. Sendo A e B conjuntos,

$a \in A$	a pertence a A, a é elemento de A
$a \notin A$	a não pertence a A
$A = B$	igualdade de conjuntos (qualquer que seja x, $x \in A$ se e só se $x \in B$)
$A \subseteq B$	A contido em B, A subconjunto de B (qualquer que seja 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 $A \subseteq B \wedge A \neq B$
$A \supset B$	A contém propriamente B
$A \neq B$	$A \not\subseteq B$ ou $B \not\subseteq A$
$\emptyset$ ou $\{\}$	conjunto vazio
$\mathcal{P}(A)$	Conjunto dos subconjuntos de A.

A **intersecção de** A **com** B, representa-se por $A \cap B$, e é o conjunto dos elementos que pertencem a ambos os conjuntos

$$A \cap B = \{x \mid x \in A \text{ e } x \in B\}$$

A **reunião de** A **com** B, representa-se por $A \cup B$, e é o conjunto dos elementos que pertencem a pelo menos um dos conjuntos.

$$A \cup B = \{x \mid x \in A \text{ ou } x \in B\}$$

O **complementar de** B **em** A, representa-se por $A \setminus B$, e é o conjunto dos elementos de A que não pertencem a B.

$$A \setminus B = \{x \mid x \in A \text{ e } x \notin B\}$$

Quando está implicíto um dado *universo* $\mathcal{U}$, chama-se **complementar de** A, e representa-se por $\overline{A}$, ao complementar de A em $\mathcal{U}$, ou seja a $\mathcal{U} \setminus A$.

Exemplo 1.1. *Mostrar* $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$ *é mostrar que* $A \setminus (B \cup C) \subseteq (A \setminus B) \cap (A \setminus C)$ *e* $A \setminus (B \cup C) \supseteq (A \setminus B) \cap (A \setminus C)$. *Para mostrar a primeira inclusão, basta mostrar que* $x \in A \setminus (B \cup C) \Rightarrow x \in (A \setminus B) \cap (A \setminus C)$, *qualquer que seja* x.

$$\begin{array}{rcll} x \in A \setminus (B \cup C) & \Longrightarrow & x \in A \wedge x \notin B \cup C & \textit{(por def. de diferença)} \\ & \Longrightarrow & x \in A \wedge (x \notin B \wedge x \notin C) & \textit{(por def. união)} \\ & \Longrightarrow & x \in A \setminus B \wedge x \in A \setminus C & \textit{(por def. de diferença)} \\ & \Longrightarrow & x \in (A \setminus B) \cap (A \setminus C) & \textit{(por def. intersecção)} \end{array}$$

Mostremos agora que $x \in (A \setminus B) \cap (A \setminus C) \Rightarrow x \in A \setminus (B \cup C)$. *Se* $x \in (A \setminus B) \cap (A \setminus C)$ *então* $x \in (A \setminus B)$ *e* $x \in (A \setminus C)$. *Logo, por definição de diferença de conjuntos,* $x \in A$ *e* $x \notin B$, *e* $x \notin C$. *Se* $x \notin B$ *e* $x \notin C$ *então* $x \notin (B \cup C)$. *E de* $x \in A$ *e* $x \notin (B \cup C)$, *segue* $x \in A \setminus (B \cup C)$

Exemplo 1.2. *Quaisquer que sejam os conjuntos* $A, B \subseteq \mathcal{U}$, *tem-se* $\overline{A \cup B} = \overline{A} \cap \overline{B}$.
Se $x \in \overline{A \cup B}$ *então* $x \notin A \cup B$. *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,

$$\begin{array}{rcll} x \in \overline{A} \cap \overline{B} & \Longrightarrow & (x \in \overline{A} \wedge x \in \overline{B}) & \textit{(por def. } \cap\textit{)} \\ & \Longrightarrow & (x \notin A \wedge x \notin B) & \textit{(por def. complementar)} \\ & \Longrightarrow & x \notin (A \cup B) & \textit{(por def. } \cup\textit{)} \\ & \Longrightarrow & x \in \overline{A \cup B} & \textit{(por def. complementar)} \end{array}$$

ou seja, $\overline{A \cup B} \supseteq \overline{A} \cap \overline{B}$.

Exercício 1.0.1. Suponha que os conjuntos indicados são subconjuntos do universo $\mathcal{U}$. Sendo A, B e C subconjuntos de $\mathcal{U}$ quaisquer, mostre cada uma das propriedades:

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

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

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

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

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

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

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

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

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

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

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

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

(u) $A \cap A = A$

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

Exemplo 1.3. *Justificar a veracidade ou falsidade das afirmações seguintes, onde $\mathbb{Z}$ denota o conjunto dos inteiros.*

1. **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 $x+1$ é um inteiro e $x+1$ é superior a x. Ou seja, dado um x qualquer, tomamos y como sendo $x+1$, satisfazemos a condição $(x \leq y \wedge x \neq y)$.

2. **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 em particular se x fosse $y+1$ então teriamos que ter $y+1 \leq y$, o que é falso.

3. **Existe um inteiro não negativo que não excede qualquer outro inteiro não negativo.**

   $$\exists_{x\in\mathbb{Z}}\, ((x \geq 0) \wedge \forall_{y\in\mathbb{Z}}\ \ ((y \geq 0) \Rightarrow x \leq y))$$

   A afirmação é verdadeira. O inteiro 0 é menor ou igual que cada um dos inteiros não negativos.

4. **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 (análoga à 2).

5. **Para todo $A \subseteq \mathbb{Z}$, tem-se $\mathcal{P}(A) = \{\emptyset\}$.**

   A afirmação é falsa, porque $\{1\}$ é um subconjunto de $\mathbb{Z}$ e $\mathcal{P}(\{1\}) = \{\emptyset, \{1\}\} \neq \{\emptyset\}$.

6. **Para todo $A \subseteq \mathbb{Z}$, se $A = \emptyset$ então $\mathcal{P}(A) = \{\emptyset\}$.**

   A afirmação é verdadeira. Só existe um subconjunto de $\mathbb{Z}$ que é vazio, e $\mathcal{P}(\emptyset) = \{\emptyset\}$. Note que $\{\emptyset\}$ é um conjunto que tem um elemento (o qual é o conjunto vazio).

7. **Tem-se $\mathcal{P}(A) = \emptyset$, para algum $A \subseteq \mathbb{Z}$.**

   $$\exists A\ \ (A \subseteq \mathbb{Z} \ \wedge \ \mathcal{P}(A) = \emptyset)$$

   A afirmação é falsa, porque qualquer que seja o subcojunto A de $\mathbb{Z}$, o conjunto vazio é um elemento de $\mathcal{P}(A)$.

8. **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 $y = x + z$. É verdade que $(x \leq y \ \vee \ y \leq x)$, quaisquer que sejam os inteiros x e y.

Como $x - y$ é inteiro, quaisquer que sejam x e y, 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 $x \leq y$.

9. **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 o 4. Por exemplo, o conjunto vazio.

10. **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 $5 \in A$.

11. **Para todo o $x \in \mathbb{Z}$, e quaisquer que sejam $A, B \subseteq \mathbb{Z}$, 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 falar em x e não em 5!).

12. **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 expressão 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 escrevemos por vezes como

$$\exists_{x\in\mathbb{Z}}\, \forall_{A\subseteq\mathbb{Z}}\, \forall_{B\subseteq\mathbb{Z}} \ \ (x \in A \setminus B)$$

que é falsa. Quaisquer que sejam os conjuntos A e B se $A = B$ então $A \setminus B = \emptyset$. Assim, se tomarmos $A = B = \{1\}$, os conjuntos A e B são subconjuntos de $\mathbb{Z}$ e não existe qualquer inteiro x tal que $x \in A \setminus B$.

13. **$2 + 2 = 4$ ou $\sqrt{2} \in \mathbb{Z}$.**

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

14. **Se** $2+2 \neq 4$ **então** $\sqrt{2} \in \mathbb{Z}$**.**

A afirmação é equivalente a "$2+2=4$ *ou* $\sqrt{2} \in \mathbb{Z}$*".*

15. **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.*

16. $\sqrt{2} \notin \mathbb{Z}$ **ou** $2+2 \neq 4$**.**

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

Nestes apontamentos, $\mathbb{N}$ representa o conjunto dos inteiros não negativos, ou seja $\mathbb{N} = \{0, 1, 2, 3, 4, 5, 6, \ldots\}$.

1.2 Indução Matemática

O método de demonstração por indução matemática (ou Indução finita) será bastante usado durante o curso pelo que é conveniente recordá-lo.

Exemplo 1.4. *Imagine* uma escada com uma infinidade de degraus*. Não é uma escada com um número enorme de degraus! Esta escada, tem sempre um degrau acima de qualquer outro que considere. Suponha que é verdade (1.1).*

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

*Se nada mais for dito, não pode concluir que "*consegue chegar ao$105^{\underline{o}}$ degrau*".*
Suponha agora não só (1.1) mas também (1.2).

*"Consigo chegar ao***13**$^{\underline{o}}$ *degrau"* (1.2)

O que pode concluir? Como consegue chegar ao$13^{\underline{o}}$ *e é verdade (1.1), então consegue chegar ao*$14^{\underline{o}}$ *. Como consegue chegar ao*$14^{\underline{o}}$ *e é verdade (1.1), então consegue chegar ao*$15^{\underline{o}}$ *. Como consegue chegar ao*$15^{\underline{o}}$ *e é verdade (1.1), então consegue chegar ao*$16^{\underline{o}}$ *. De (1.1) e (1.2), conclui-se (1.3).*

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

Do mesmo modo, se supuser (1.1) e (1.4)

"Consigo chegar ao $\mathbf{1}^{\underline{o}}$ *degrau".* (1.4)

então conclui-se (1.5).

"Consegue chegar ao **n***-ésimo degrau, qualquer que seja* $n \geq 1$*."* (1.5)

Exemplo 1.5. *Seja* A_n *a área dum 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. *Mostremos que* qualquer que seja $k \geq 1$, **se** A_k for da forma $3p+1$ para algum inteiro não negativo p, **então** $A_{k+1} = 3q+1$ para algum inteiro não negativo q.

   Ora, se $A_k = 3p+1$*, então* $A_{k+1} = (2^{k+1})^2 = 4(2^k)^2 = 4A_k = 4(3p+1) = 3(4p+1)+1$*. Ou seja,* $A_{k+1} = 3q+1$ *para algum inteiro não negativo, já que se* $p \in \mathbb{Z}$ *e* $p \geq 0$ *então* $4p+1 \in \mathbb{Z}$ *e* $4p+1 \geq 0$*.*

2. *Sabemos também que o resto da divisão de 4 (isto é, de* A_1*) por 3 é 1.*

O que podemos concluir sobre A_1*,* A_2*,* A_3*,... ? Por (2), sabemos que o resto da divisão de* A_1 *por 3 é 1.*

De (2) e (1), 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, concluimos, por (1), que o resto da divisão de* A_3 *por 3 é 1.*
Como o resto da divisão de A_3 *por 3 é 1, concluimos, por (1), que o resto da divisão de* A_4 *por 3 é 1.*
Como o resto da divisão de A_4 *por 3 é 1, concluimos, por (1), que o resto da divisão de* A_5 *por 3 é 1.*
Como o resto da divisão de A_5 *por 3 é 1, concluimos, por (1), que o resto da divisão de* A_6 *por 3 é 1.*
⋮

Do mesmo modo que conseguimos mostrar que A_6 *dá resto 1 quando dividido por 3, podemos, dado um* $n \geq 1$*, apresentar a dedução de que* A_n *dá resto 1 quando dividido por 3.*

Assim, de (2) e (1), podemos concluir que qualquer que seja $n \geq 1$*, a área do quadrado de lado* 2^n *excede numa unidade um múltiplo de 3.*

Proposição 1.1. *(***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) $\forall k \in \mathbb{N}\ [\ P(k) \Rightarrow P(k+1)\ ]$*, isto é para todo* $k \in \mathbb{N}$*, se* $P(k)$ *então* $P(k+1)$*;*

$P(0)$ 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**.

Proposição 1.2. *(**Versão geral do Princípio de Indução**)*
Seja $n_0 \in \mathbb{Z}$, e suponha 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) $\forall k \geq n_0 \ [\ P(k) \Rightarrow P(k+1)\ ]$, *isto é, para todo inteiro $k \geq n_0$, se $P(k)$ então $P(k+1)$;*

Exemplo 1.6. *Vamos mostrar que qualquer que seja $n \geq 1$ e quaisquer que sejam $a, b \in \mathbb{R}^+$, se tem $(a+b)^n < 2^n(a^n + b^n)$.*
Podemos concluir isso se mostrarmos que as duas condições de aplicabilidade do princípio de indução matemática se verificam neste caso.

(i) *Para n=1, $(a+b)^1 = a+b < 2(a+b) = 2^1(a^1+b^1)$, porque $a, b \in \mathbb{R}^+$.*

(ii) *Suponhamos agora que para um dado $k \geq 1$ 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

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

Como

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

para concluir que $(a+b)^{k+1} < 2^{k+1}(a^{k+1}+b^{k+1})$ vamos mostrar que

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

quaisquer que sejam a, b reais positivos. Tem-se

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

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

Exercício 1.0.2. Aplique o princípio de indução matemática para resolver os problemas seguintes.

1. Mostre que $|sen(nx)| \leq n|sen(x)|$, para todo $x \in \mathbb{R}$ e todo $n \in \mathbb{N}$.
2. Mostre que $4^n + 15n - 1$ é múltiplo de 9, para todo $n \geq 1$.
3. ** Seja $(x_n)_{n\geq 1}$ uma sucessão de números reais, tal que $x_1 \neq 0$, e para qualquer inteiro $p \geq 3$ se verifica

$$\sum_{i=1}^{p-1} x_i^2 \sum_{i=2}^{p} x_i^2 = \left(\sum_{i=1}^{p-1} x_i x_{i+1}\right)^2 \tag{1.6}$$

   (a) Mostre que é necessário que x_1, x_2 e x_3 sejam termos consecutivos duma progressão geométrica. Sugestão: suponha que $x_2 = \alpha x_1$ e $x_3 = \beta x_1$ e mostre que para se verificar (1.6) para $p = 3$ tem que ser $\beta = \alpha^2$.

   (b) Mostre que nas condições do problema, se x_1, x_2, x_3, $\ldots x_k$ são termos consecutivos duma progressão geométrica de razão r então também x_{k+1} é termo dessa progressão, e $x_{k+1} = rx_k$, qualquer que seja $k \geq 3$.

   (c) Conclua que a sucessão $(x_n)_n$ é uma progressão geométrica.

Exemplo 1.7. *Considere o seguinte algoritmo (imagine que Y é uma caixa).*

1. *Coloque em Y o valor 0.*
2. *Escolha um inteiro (vamos referi-lo por X).*
3. *Se $X = 0$ então faça* **7.**
4. *Se $X > 0$ então volte a* **2.**
5. *Substitua o valor em Y pela soma do valor que lá estava com dois.*
6. *Volte a* **2.**
7. *Indique o valor que guarda em Y, e páre.*

O valor que está guardado em Y quando está a executar **2.** *pela $2^{\underline{a}}$ vez é 0 se o primeiro valor escolhido em* **2.** *for positivo, e é 2 se tal valor for negativo.*
Quando está a executar **2.** *pela $3^{\underline{a}}$ vez, o valor em Y é 4 se ambos os valores escolhidos anteriormente forem negativos, é 2 se um for negativo e o outro positivo, e é 0 se ambos forem positivos.*
Suponhamos que para um dado $k \geq 1$, quando se se está a executar **2.** *pela $k^{\underline{a}}$ vez, o valor em Y é o dobro do número de inteiros negativos nos $k - 1$ valores escolhidos anteriormente.*

Então, por análise do algoritmo, concluimos que se se estiver a executar **2.** *pela*$(k+1)^{\underline{a}}$ *vez, o valor em Y é igual ao valor anterior se tivermos dado mais um número positivo, ou foi incrementado de duas unidades, se tivermos dado mais um número negativo. Portanto, se quando se estiver a executar* **2.** *pela*$k^{\underline{a}}$ *vez, o valor em Y é o dobro do número de inteiros negativos nos* $k-1$ *valores escolhidos anteriormente, então quando se estiver a executar* **2.** *pela*$(k+1)^{\underline{a}}$ *vez, valor em Y é o dobro do número de inteiros negativos nos k valores escolhidos anteriormente.*

Consequentemente, por indução matemática, podemos concluir que o valor em Y quando executa **7.** *é o dobro do número de inteiros negativos nos valores escolhidos.*

1.2.1 Erros frequentes

Exemplo 1.8. *Considere a seguinte demonstração*[1] *de que "*entre dois inteiros consecutivos existem 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* $k+1$ *existirem 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 existem uma infinidade de inteiros.* □

Como entre dois inteiros consecutivos não há qualquer outro inteiro, a prova tem que estar errada. O erro está na conclusão "precipitada". Não se mostrou que existia um intervalo $[k, k+1]$ *que tinha uma infinidade de inteiros. Apenas se mostrou a condição (ii) do princípio de indução. A condição (i) não foi provada, nem se pode provar!*

Este exemplo serve para ilustrar o facto de certas propriedades, não observáveis, poderem ser hereditárias[2]*.*

Exemplo 1.9. *A demonstração seguinte está obviamente errada dado que permite concluir que "todos os cavalos são brancos"*[3]*.*

[1] Claramente, está errada!

[2] Hereditárias, no sentido de se um dado inteiro as satisfizer, também o inteiro seguinte as satisfaz.

[3] E, todos sabemos que há cavalos doutras 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 nessa situação.*

(i) *Como pelo menos deve estar um cavalo branco na cerca, podemos afirmar que se só existir um cavalo na cerca é 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 todos os demais são brancos. Consideramos agora uma cerca onde estão $k+1$ cavalos sendo branco pelo menos um deles. Retiremos um cavalo da cerca deixando ficar o 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 cavalos que está na cerca (é indiferente qual se retira porque de facto todos são brancos), e voltemos a colocar o que retirámos primeiramente. Mais uma vez, pela hipótese, podemos concluir que todos os cavalos que estão agora na cerca são brancos.*

Assim, pelo princípio de indução matemática segue a validade da proposição que queriamos mostrar. □

Sabemos que "se existir algum cavalo branco numa cerca", também lá podem estar cavalos doutras cores. Logo, a afirmação que "mostrámos" por indução é falsa. Mas, se o sistema dedutivo for consistente, não podemos deduzir proposições falsas, pelo que há que encontrar um erro na prova dada.
Sabemos que se estiverem dois cavalos numa cerca e um deles for branco, então isso 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).

Note que 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 conseguimos mostrar que "todos os cavalos são brancos" porque $P(1) \not\Rightarrow P(2)$.

Exemplo 1.10. *Neste exemplo, a prova está errada, mas o resultado é válido. Pretendemos mostrar* **que qualquer que seja $n \geq 1$, o quadrado de lado 2^n pode ser coberto por peças da forma seguinte, ficando apenas um quadrado 1×1 por cobrir**.

A propriedade verifica-se para $n = 1$, por exemplo:

Para concluir, por indução, que a propriedade é válida para todo $n \geq 1$, resta mostrar que qualquer que seja $k \geq 1$, se os quadrados de lado 2^k admitem a cobertura descrita, então os quadrados de lado 2^{k+1} também admitem.

Dado um quadrado de lado 2^{k+1} podemos dividi-lo em quatro quadrados de lado 2^k como se ilustra na figura abaixo. Por hipótese, cada quadrado de lado de 2^k pode ser coberto por peças da forma descrita acima. Fazemos para cada um dos quatro uma tal cobertura de forma a ficarem a descoberto os quadrados 1×1 indicados na figura. Verificamos que os três quadrados 1×1 (ao centro) podem ser cobertos por uma peça, pelo que só fica um quadrado 1×1 por cobrir.

Esta prova não está correcta! *Porquê? Na demonstração supusemos que o quadrado por cobrir, podia estar num canto, mas o quadrado que deixámos depois a descoberto não ficou*

num canto. No entanto, podemos corrigir a prova, mostrando um resultado mais forte do que o enunciado inicialmente... "a cobertura pode ser tal que o quadrado 1×1 *fica num canto". Para isso, basta notar que caso* $n = 1$*, o quadrado por cobrir está num canto, e que para mostrar que* $\forall n \ (P(n) \Rightarrow P(n+1))$ *podemos fazer o seguinte.*

1.2.2 Indução forte

Exemplo 1.11. *Dado um conjunto finito* Σ *de símbolos, diz-se* palavra *qualquer sequência finita de símbolos de* Σ*. Por exemplo, se* $\Sigma = \{\mathtt{a}, \mathtt{b}\}$*, as palavras são sequências de* `a`*'s ou* `b`*'s.*

Seja L *o conjunto de todas as palavras constituídas por* `a`*'s ou* `b`*'s formadas por aplicação das duas regras seguintes.*

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

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

As palavras `aaabbaaa` *e* `aaabbaaabbaaabbaaa` *são palavras de* L*,*

Denote-se o número de `a`*'s da palavra* α *por* $n_a(\alpha)$*.*

- **Para todo** p **positivo e múltiplo de 3, existe** $\gamma \in L$ **tal que** $n_a(\gamma) = p$**.**

  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 $n_a(\gamma') = p - 3$*, podemos concluir que para* p *também existe* $\gamma \in L$ *tal que* $n_a(\gamma) = p$*. De facto, basta tomar* γ *como sendo* $\mathtt{aabb}\gamma'$*, já que, por (r2), tem-se* $\mathtt{aabb}\gamma' \in L$*.*

  Assim, **por indução matemática***, concluimos que qualquer que seja* $n \geq 1$*, existe* $\gamma \in L$ *tal que* $n_a(\gamma) = 3n$*.* □

- **Qualquer que seja** $\alpha \in L$**, tem-se** $n_a(\alpha)$ **é positivo e múltiplo de 3.**

  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 $1^{\underline{o}}$ *caso, concluimos que* $n_a(\gamma)$ *é múltiplo de* 3*.*

  No $2^{\underline{o}}$ *caso, como* $n_a(\gamma) = n_a(\gamma_1) + n_a(\gamma_2)$*, podemos concluir que* $n_a(\gamma)$ *é múltiplo de* 3 se supusermos que $n_a(\gamma_1)$ e $n_a(\gamma_2)$ são múltiplos de 3... *ou ainda,* se supusermos que $n_a(\alpha)$ é múltiplo de 3, qualquer que seja $\alpha \in L$ com menor número de símbolos do que γ.

- **Qual é a forma geral duma palavra de L?**

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

onde $\mathtt{aaa}(\mathtt{bbaaa})^n$ *denota* $\mathtt{aa}\underbrace{\mathtt{bbaaa}\ldots\mathtt{bbaaa}}_{n\ vezes}$.

Como exercício, mostre, por indução sobre n, que:

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

Vamos justificar*:* $L \subseteq \{\mathtt{aaa}\} \cup \{\mathtt{aaa}(\mathtt{bbaaa})^n \mid n \geq 1\}$.

Se $\gamma \in L$ *então* $\gamma = \mathtt{aaa}$ *ou existem* $\gamma_1, \gamma_2 \in L$ *tais que* $\gamma = \gamma_1\mathtt{bb}\gamma_2$.

No2^{o} *caso,* γ *tem pelo menos quatro símbolos.* Suponhamos que para qualquer palavra $\alpha \in L$ com menos símbolos do que γ, existe $m \geq 0$ tal que α é da forma

$$\mathtt{aa}\underbrace{\mathtt{bbaaa}\ldots\mathtt{bbaaa}}_{m\ vezes}$$

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

$$\gamma = \mathtt{aaa}\underbrace{\mathtt{bbaaa}\ldots\mathtt{bbaaa}}_{m_1\ vezes}\mathtt{bbaa}\underbrace{\mathtt{bbaaa}\ldots\mathtt{bbaaa}}_{m_2\ vezes}$$

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

A versão seguinte do princípio de indução suporta esta justificação.

Proposição 1.3. (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* $P(k+1)$.

Exemplo 1.12. *Diz-se* primo *qualquer inteiro x que admite dois e apenas dois divisores positivos (*1 *e x).* **Vamos mostrar, por indução, que qualquer inteiro** $n \geq 2$ **ou é primo ou pode escrever-se como um produto de primos.** *Para isso basta mostrar que as duas condições (i) e (ii) de aplicabilidade do princípio de indução se verificam.*

(i) *Pela definição de primo, tem-se* 2 *é primo.*

(ii) *Seja* $x \in \mathbb{N}$ *tal que* $x > 2$. *Supomos como hipótese de indução que todo* $y \in \mathbb{N}$, *tal que* $2 \leq y < x$, *ou é primo ou é produto de primos, para concluir que então* x *é primo ou produto de primos. De facto, se* x *não for primo, existem* $x_1, x_2 \in \mathbb{N} \setminus \{0, 1\}$ *tais que* $x = x_1 x_2$. *Como* $2 \leq x_1 < x$, *e* $2 \leq x_2 < x$, *concluimos pela hipótese de indução, que* x_1 *é primo ou produto de primos, e* x_2 *é primo ou produto de primos.*

Analisando os quatro casos (a) x_1 *e* x_2 *primos, (b)* x_1 *e* x_2 *produtos de primos, (c)* x_1 *primo e* x_2 *produto de primos, e (d)* x_1 *produto de primos e* x_2 *primo, concluimos que se* $x = x_1 x_2$ *então* x *pode escrever-se como produto de primos.*

Logo, se x *não é primo então* x *é um produto de primos, ou seja,* x *é primo ou produto de primos.* □

Exemplo 1.13. *Seja "***QS***" (é conhecido por* quick-sort*) o algoritmo seguinte.*

0. *Supõe-se dada uma sequência finita de inteiros (possivelmente com zero elementos).*

1. *Se a sequência não tem qualquer elemento então indicar como resultado a própria sequência.*

2.1 *Senão seja* X *o primeiro elemento na sequência dada.*

2.2 *Separar os* restantes *elementos da sequência formando a* subsequência dos menores do que X, *e a* subsequência dos maiores ou iguais que X.

2.3 *Aplicar "QS" a cada uma das subsequências: sejam* $R_<$ *e* $R_\geq$ *as sequências resultantes (respectivamente).*

2.4. *Indicar como resultado os elementos de* $R_<$, *seguidos de* X, *seguidos dos elementos de* $R_\geq$.

Vamos mostrar que o algoritmo "QS" ordena por ordem crescente a sequência finita de inteiros dada. A prova vai ser por indução sobre o número de elementos na sequência dada.
Vamos mostrar que as condições (i) e (ii) do princípio de indução estão satisfeitas, para concluir que qualquer que seja a sequência finita de inteiros dada, "QS" indica-a por ordem crescente.

(i) *Se a sequência dada não tem elementos, então podemos dizer que a sequência resultante, que é igual à dada (por* **1.***) está ordenada por ordem crescente.*

(ii) *Seja S uma qualquer sequência de inteiros, e seja n ($n \geq 1$ fixo) o seu número de elementos. Suponhamos agora, como hipótese de indução, que quando aplicamos "QS" a uma qualquer sequência W de inteiros* cujo número de elementos é menor do que n, *obtemos os elementos de W por ordem crescente.*

Vamos então mostrar que "QS" quando aplicado a S, a ordena por ordem crescente.

Como S tem elementos, aplicam-se as instruções 2.1–2.4. Uma vez que nas subsequências formadas em **2.2** *não entra o primeiro elemento de S, cada uma delas tem menos elementos do que S. Assim, por hipótese de indução, "QS" quando aplicado a cada uma dessas subsequências, ordena-a por ordem crescente. Então, as sequências referidas por $R_<$ e $R_\geq$ (em* **2.3***) são respectivamente as sequências de elementos menores do que X e maiores ou iguais a X, ordenadas por ordem crescente.*

Donde, o resultado indicado em **2.4** *não é mais do que a sequência S ordenada por ordem crescente.* □

Proposição 1.4. *Os princípios de indução forte e fraca são equivalentes, ou seja se existir uma prova de $\forall n \in \mathbb{N} \;\; P(n)$ por indução fraca então também existe uma prova por indução forte. E vice-versa, se existir uma prova por indução forte então existe uma prova por indução fraca.*

1.2.3 Outras formulações do princípio de indução

Proposição 1.5. *Seja $A = \{n \in \mathbb{N} \mid P(n)\}$, onde $P(n)$ denota "n satisfaz a propriedade P". Se A satisfizer as condições (i) e (ii) seguintes, então $A = \mathbb{N}$.*

$$\begin{array}{ll} (i) & 0 \in A \\ (ii) & \forall k \in \mathbb{N} \;\; (k \in A \Rightarrow k+1 \in A) \end{array}$$

Proposição 1.6. *Seja $A = \{n \in \mathbb{N} \mid P(n)\}$, onde $P(n)$ denota "n satisfaz a propriedade P". Se A satisfizer as condições (i) e (ii) seguintes, então $A = \mathbb{N}$.*

$$\begin{array}{ll} (i) & 0 \in A \\ (ii) & \forall k{\in}\mathbb{N} \;\; [\,(\forall i \leq k \;\; i{\in}A) \Rightarrow k+1 \in A\,] \end{array}$$

Capítulo 2

Linguagens Formais e Modelos de Computação

2.1 Alfabeto, palavra e linguagem

Designamos por **alfabeto** qualquer conjunto finito e não vazio. Os elementos dum alfabeto dizem-se **símbolos**. Uma sequência finita de símbolos dum alfabeto Σ diz-se **palavra** (ou ainda, **frase** ou **sequência**) de alfabeto Σ. As palavras podem ter zero ou mais símbolos. A **palavra vazia**, que convencionamos representar por ε, é a palavra com zero símbolos. O número de símbolos que constituem a palavra, é o **comprimento** da palavra. O comprimento duma palavra α 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 2.1. *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 dum dicionário de Língua Portuguesa é uma* linguagem *dum 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 sintacticamente correctas (de acordo com as regras gramaticais de Português) é outra linguagem. Usando a terminologia anterior, pode-se dizer 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 seguintes.*

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

Em algumas linguagens (por exemplo, na língua Portuguesa escrita), há que atender não só à boa formação sob o ponto de vista da **sintaxe**, *mas também da* **semântica**. *Talvez em Português Literário possa encontrar* `o livro leu o gato`. *Para que fique claro, nesta disciplina, estamos interessados em aspectos da sintaxe mais do que nos da semântica.*

Exemplo 2.2. *As palavras de alfabeto* $\{\mathtt{a},\mathtt{b}\}$ *que têm comprimento não superior a dois são* ε, `a`, `b`, `aa`, `ab`, `ba`, *e* `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 por* $\{x \mid x$ *é palavra de alfabeto* $\{\mathtt{a},\mathtt{b}\}$ *tal que* $|x| \leq 2\}$.

Exemplo 2.3. *Seja* L *a linguagem de alfabeto* $\Sigma = \{\mathtt{p},\wedge,\vee,\tilde{},\Rightarrow,(,)\}$ *assim definida indutivamente: as palavras de* L *são que se podem obter usando as regras (i) e (ii).*

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

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

Alguns exemplos de palavras de L*:*

$$(\mathtt{p}\wedge\mathtt{p})$$
$$((\mathtt{p}\wedge\mathtt{p})\vee\mathtt{p})$$
$$(((\mathtt{p}\wedge\mathtt{p})\vee\mathtt{p})\Rightarrow(\tilde{}\,\mathtt{p}))$$
$$((((\mathtt{p}\wedge\mathtt{p})\vee\mathtt{p})\Rightarrow(\tilde{}\,\mathtt{p}))\Rightarrow((\tilde{}\,\mathtt{p})\vee\mathtt{p}))$$

2.2 Concatenação de palavras

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** $(\cdot)$ de α com β, e representa-se $\alpha \cdot \beta$ por $\alpha\beta$.
A concatenação $\cdot$ é uma operação binária em $\Sigma^\star$ que é associativa e tem elemento neutro:

- é associativa $(\alpha\beta)\gamma = \alpha(\beta\gamma)$
- a palavra vazia é identidade: $\varepsilon\alpha = \alpha\varepsilon = x$

Podemos então dizer que $(\Sigma^\star, \cdot, \epsilon)$ é um monóide.
Note que normalmente não usamos o símbolo que representa o operador de concatenação.

Exemplo 2.4. *A concatenação da palavra* `flor` *com a palavra* `bela` *é a palavra* `florbela`*. A concatenação de* `bela` *com* `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. A palavra ϵ é uma subpalavra de qualquer palavra.

Exercício 2.0.3. Sendo $\Sigma = \{0, 1\}$ e $x = 010110$. Indique-se as seguintes palavras são subpalavras de x: 011, 10 e 111.

Exercício 2.0.4. Quais os prefixos de 010111 ? E os sufixos?

Exemplo 2.5. *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 $(\alpha) \in L$*

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

A palavra $\mathbf{x^n}$ é a concatenação de n cópias da palavra x. Define-se indutivamente por:

$$x^0 = \varepsilon$$
$$x^{n+1} = x^n x$$

Exercício 2.0.5. Sendo $w = aaba$ determine w^0, w^1, w^2, w^3 e w^4

Exercício 2.0.6. Prove que, para todo $n \geq 0$, $|x^n| = n|x|$.

Em particular temos que $\mathbf{a^n}$ é a palavra constituída por n símbolos a. Por exemplo,
$a^5 = aaaaa$, $a^1 = a$ e $a^0 = \epsilon$.
Podemos definir a^n indutivamente:

$$a^0 = \epsilon$$
$$a^{n+1} = a^n a$$

A **palavra inversa** de α, denota-se por α^R e pode-se definir por indução no comprimento de α:

1. Se $|\alpha| = 0$ então $\alpha^R = \alpha = \epsilon$
2. Se $|\alpha| = n + 1$, então $\alpha = \beta a$ $(a \in \Sigma)$ e $\alpha^R = a\beta^R$

Exercício 2.1. *Determine as inversas de: aabaab, bbabb, b* $\diamond$

Exercício 2.2. *Mostre por indução no comprimento de* β *que* $(\alpha\beta)^R = \beta^R\alpha^R$. $\diamond$

2.3 Concatenação de linguagens

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

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

Por exemplo,

$$\{a, ab\}\{b, ba\} = \{ab, aba, abb, abba\}$$

Normalmente $LM \neq ML$.
As potências $\mathbf{L^n}$ de uma linguagem L são definidas indutivamente por:

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^{n+1} &= LL^n, \quad \text{para } n \geq 1 \end{aligned}$$

ou por $L^n = \{x_1x_2 \ldots x_n | x_i \in L, 1 \leq i \leq n\}$
Isto é, a linguagem das palavras obtidas por justaposição de n palavras de L.

Exemplo 2.6. *Seja* $L = \{ab, aab\}$ *então,*

$$\begin{aligned} \{ab, aab\}^0 &= \{\varepsilon\} \\ \{ab, aab\}^1 &= \{\mathtt{ab}, \mathtt{aab}\} \\ \{ab, aab\}^2 &= \{\mathtt{abab}, \mathtt{abaab}, \mathtt{aabab}, \mathtt{aabaab}\} \\ \{ab, aab\}^3 &= \{ababab, ababaab, abaabab, aababab,\ abaabaab, aababaab, aabaabab, aabaabaab\} \end{aligned}$$

Exercício 2.2.1. Mostrar a equivalência das duas definições dadas para L^n, isto é que para todo $n \in \mathbb{N}$, $(L^0 = \{\varepsilon\} \ \wedge \ L^n = LL^{n-1}) \ \Leftrightarrow \ L^n = \{w_1 \ldots w_n \mid w_i \in L,\ 1 \leq i \leq n\}$

Exemplo 2.7. *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

$$\{w \in \Sigma^\star \mid \ |w| = n\}, \text{ com } n \in \mathbb{N}$$

ou

$$\{w \in \Sigma^\star \mid \ |w| = n \text{ e } n \in \mathbb{N}\}$$

pois, $\{w \in \Sigma^\star \mid \ |w| = n\}$ é o conjunto das palavras que têm um dado comprimento (o qual foi denotado por n), enquanto que $\{w \in \Sigma^\star \mid \ |w| = n \text{ e } n \in \mathbb{N}\}$ é $\Sigma^\star$.

Exemplo 2.8. *Seja* $L = \{\mathtt{a}, \mathtt{ab}, \mathtt{bb}\}$ *então*

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

2.4 Fecho de Kleene duma linguagem

O **fecho de Kleene** de uma linguagem L é a reunião de todas as potências finitas de L

$$L^\star = L^0 \cup L^1 \cup L^2 \cup L^3 \ldots = \cup_{n \geq 0} L^n$$

ou, equivalentemente, é o conjunto das palavras que se podem formar por justaposição de zero ou mais palavras de L,

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

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

Exemplo 2.9.

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

Exercício 2.2.2. Para concluir que $\{\mathtt{000}, \mathtt{00000}\}^\star = \{\varepsilon, \mathtt{000}, \mathtt{00000}, \mathtt{000000}\} \cup \{\mathtt{0}^n \mid n \geq 8\}$ suponha que pretendia selar uma carta e só havia disponíveis selos de cinco e de três escudos. Se não existisse qualquer limite ao número de selos que podia usar, como é que obtinha as quantias seguintes 8$00, 9$00, 10$00, 11$00, 12$00, 13$00, 14$00, 15$00, 16$00 ...? Dada uma quantia na sequência, tente encontrar uma regra que lhe permita obter a quantia seguinte, trocando o menor número de selos que conseguir.

Exercício 2.3. *Seja* Σ *o alfabeto* $\{0, 1\}$. *Sendo* $A = \{10, 11\}$ *e* $B = \{00, 1\}$, *determine:* $A \cup B$, AB,BA, A^3, $A^\star$ $\diamond$

Exercício 2.4. *Seja* `aaaba` *uma palavra de alfabeto* $\{\mathtt{a}, \mathtt{b}\}$. *A que linguagens pertence:*

a) $\{\mathtt{a}, \mathtt{b}\}^*$

b) $\{\mathtt{aaa}, \mathtt{bab}\}\{\mathtt{ba}, \mathtt{bb}\}$

c) $\{\mathtt{aaa}\}^*\{\mathtt{b}\}^*\{\mathtt{a}\}$

d) $\{\mathtt{a}\}^*\{\mathtt{b}\}^*\{\mathtt{a}\}^*$

e) $\{\mathtt{aa}\}^*\{\mathtt{a}\}^*\{\mathtt{a}, \mathtt{ba}, \mathtt{bb}, \varepsilon\}$

$\diamond$

Lema 2.1. *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[1] 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, 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 2.2. *Sejam* L_1 *e* L_2 *linguagens de alfabeto* Σ. *Se* $\varepsilon \in L_1$ *então* $L_2 \subseteq L_1L_2$, *e se* $\varepsilon \in L_2$ *então* $L_1 \subseteq L_1L_2$.

[1] "**NEM TODAS AS PROVAS SÃO ILEGÍVEIS ou DIFÍCEIS!**"

Exercício 2.4.1. Partindo da definição da concatenação de duas linguagens, mostre o Lema 2.2.

Proposição 2.1. *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) e (11) ficando as restantes ao cuidado do leitor.

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

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 dum 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 duma 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 2.1, 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 2.1, 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 2.2, 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, $R \cup S \subseteq R^\star S^\star$ □

Capítulo 3

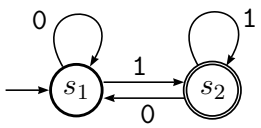
Autómatos finitos

3.1 Autómatos Finitos Determinísticos (AFDs)

O comportamento de muitos sistemas reais podem ser descritos através de um número finito de configurações e de transições entre elas. Exemplos: elevadores, circuitos electrónicos, diversos sistemas biológicos, dinheiro electrónico, ...

Um **autómato finito determinístico** é um modelo matemático de um sistema cujo comportamento pode ser descrito como uma sequência simples, discreta e linear 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. Cada **transição** é produzida por um *estimulo* ou *entrada*. Considera-se que cada estímulo (**símbolo de entrada**) determina um e um só estado posterior, ou seja tem um efeito **determinístico**: o estado seguinte é uma função do estado actual e do símbolo de entrada. A sequência de configurações anteriores a um dado estímulo constitui a história do sistema.

Exemplo 3.1. *O diagrama seguinte representa um AFD que tem dois estados,* s_0 *e* s_1*. As* **transições** *representadas pelas "setas" (a que se chama* **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*.*

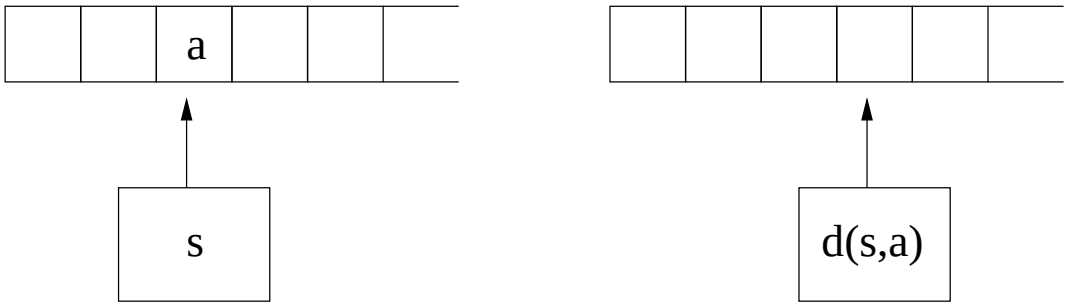


Definição 3.1. *Um* **autómato finito determinístico** *é um quinteto* $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ *onde:*

- 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**
- δ *uma função de* $S \times \Sigma$ *em* S*, designada por* **função de transição**.

3.1.1 Autómato como uma máquina finita

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, na qual está escrita uma sequência de símbolos de um alfabeto. A máquina tem uma cabeça de leitura (colocada inicialmente de forma a ler o símbolo mais à esquerda na sequência) e um mecanismo de transição de estados.



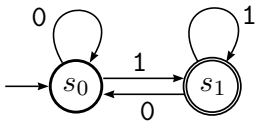
O seu funcionamento pode ser descrito do seguinte modo: num dado estado s (cabeça de leitura numa dada posição) lê o símbolo na fita, seja a, passa ao estado $\delta(s, a)$ e move a cabeça de leitura para a posição seguinte (à direita).
Diz-se que a máquina **aceita** (ou **reconhece**) a sequência dada na fita, se quando acabar de a ler estiver num estado final. A linguagem **aceite** ou **reconhecida** pelo autómato $\mathcal{A}$ é o conjunto das palavras em $\Sigma^\star$ que são aceites pelo autómato o qual denotamos por $L(\mathcal{A})$. **Uma linguagem L é aceite pelo autómato $\mathcal{A}$** se e só se $L = L(\mathcal{A})$. A **função de transição** é uma relação binária de $S \times \Sigma$ em S, pode representar-se por um conjunto de pares $((s, a), s')$, com $s, s' \in S$, $a \in \Sigma$. Para facilitar a notação usamos ternos (s, a, s').

Exemplo 3.2. *Seja* $A = (\{s_0, s_1\}, \{0, 1\}, \delta, s_0, \{s_1\})$ *um autómato finito determinístico em que,*

$$\delta = \{(s_0, 0, s_0), (s_0, 1, s_1), (s_1, 1, s_1), (s_1, 0, s_0)\} \quad ou$$

δ	0	1
$\rightarrow s_0$	s_0	s_1
$\star s_1$	s_0	s_1

A um autómato finito fica naturalmente associado um multigrafo dirigido com símbolos associados aos arcos. Os vértices correspondem aos estados do autómato; à transição do estado s para o estado s' pelo símbolo de entrada a, corresponde um arco etiquetado por a do nó associado a s para o nó associado a s'. Esse multigrafo designa-se por **diagrama de transição**. Para o exemplo anterior, podemos representar o diagrama de transição esquematicamente do modo seguinte.



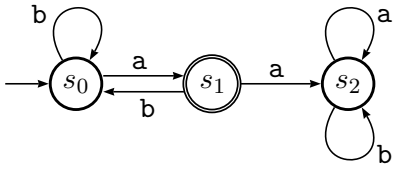
Por convenção, os estados finais são representados por duas circunferências, e o estado inicial é apontado por uma seta.

Exemplo 3.3. *Analise cada um dos exemplos seguintes.*

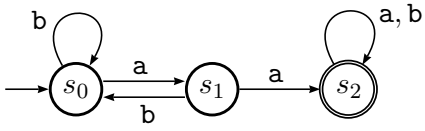
a) A linguagem de alfabeto $\{\mathtt{a}, \mathtt{b}\}$ *constituída pelas palavras que terminam em* a *e não têm* a*'s consecutivos, i.e.,*

$$\{x \in \{\mathtt{a}, \mathtt{b}\} \mid x \textit{ termina em } \mathtt{a} \textit{ e não têm } \mathtt{a}\textit{'s consecutivos}\}$$

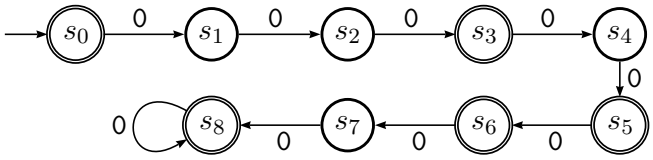
é a linguagem reconhecida pelo autómato seguinte.



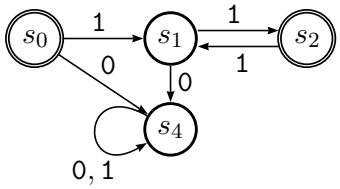
b) A linguagem $\{x \mid x \in \{\mathtt{a}, \mathtt{b}\}^\star$ *e tem* aa *como subpalavra*$\}$*, é a reconhecida pelo autómato*



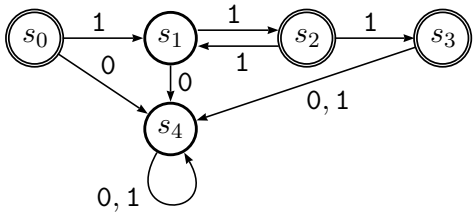
c) O autómato finito determinístico que tem menor número de estados e reconhece $\{\mathtt{000}, \mathtt{00000}\}^\star$ *de alfabeto* $\{\mathtt{0}\}$ *é:*



d) A linguagem $\{(11)^n \mid n \geq 0\}$ *é aceite pelo autómato*



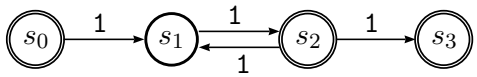
e) A linguagem $\{(\mathtt{11})^n \mid n \geq 0\} \cup \{(\mathtt{11})^n\mathtt{0} \mid n \geq 1\}$ *é a linguagem reconhecida, por exemplo, pelo autómato*



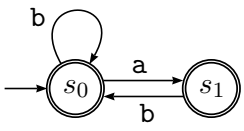
3.1.2 Autómatos que encravam

Quando a função de transição δ não é total, isto é quando δ não estiver definida para alguns pares $(s, a) \in S \times \Sigma$, pode acontecer que o autómato **encrave** 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 façam o autómato encravar são consideradas como **não aceites**.

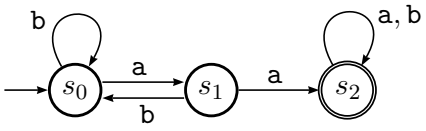
Exemplo 3.4. *A linguagem* $\{(\mathtt{11})^n \mid n \geq 0\} \cup \{(\mathtt{11})^n\mathtt{0} \mid n \geq 1\}$ *é a linguagem reconhecida, por exemplo, pelo autómato*



Exemplo 3.5. *A linguagem das palavras em* $\{\mathtt{a}, \mathtt{b}\}^\star$ *que não têm* `aa` *como subpalavra é a linguagem reconhecida pelo autómato seguinte.*



Este autómato encrava se a palavra tiver aa *como subpalavra.*
Se quiséssemos representar o autómato que reconhecia $\{x\texttt{aa}y \mid x, y \in \{\texttt{a},\texttt{b}\}^\star\}$*, isto é a linguagem das palavras que têm* aa *como subpalavra, então o "estado" que anteriormente omitimos é agora relevante.*



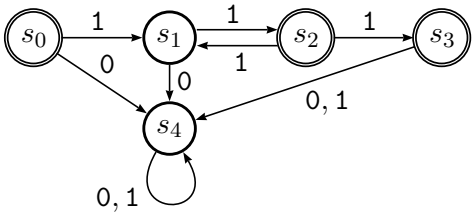
3.1.3 Extensão da função de transição a palavras

Dado $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$, seja a função $\widehat{\delta} : S \times \Sigma^\star \longrightarrow S$ definida a partir de δ por indução no comprimento de x, $|x|$:

$$\begin{aligned} \widehat{\delta}(s, \varepsilon) &= s \\ \widehat{\delta}(s, xa) &= \delta(\widehat{\delta}(s, x), a) \end{aligned}$$

Notar que $\widehat{\delta}(s, a) = \delta(s, a)$!
Associamos a cada estado o conjunto de palavras que ele "memoriza".
$L(s) = \{x \in \Sigma^\star \mid \widehat{\delta}(s_0, x) = s\}$

Exemplo 3.6. *Para o exemplo 3.3.e), cujo autómato é*



Temos:

s_0 *palavra vazia,* ε

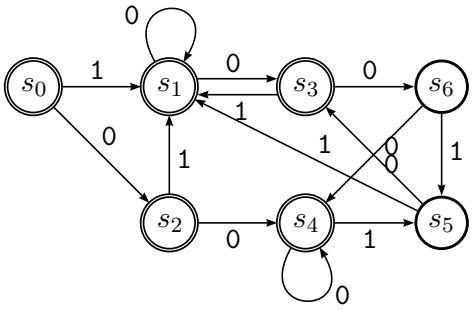
s_1 *número ímpar de* 1*'s consecutivos*

s_2 *número par de* 1*'s consecutivos*

s_3 *número par de* 1*'s consecutivos, seguido de um* 0

s_4 *contém algum* 0 *não antecedido de um número par de* 1*'s ou seguido de algum* 1

Exemplo 3.7. *O AFD mínimo que reconhece* $\{x \mid x \in \{\texttt{0},\texttt{1}\}^\star$ *e* x *não termina em* 100 *nem em* 001$\}$ *é o representado abaixo; qualquer AFD que reconheça esta linguagem tem maior número de estados do que este.*



Como justificar a necessidade de cada uma destas transições?

- *Se a palavra começar por* `1`*, e o autómato consumisse* `1` *no estado inicial e se mantivesse no mesmo estado, então não memorizava o facto de esse* `1` *poder ser o início de* `100`*. Era como se esse primeiro* `1` *não fosse relevante...*
- *Se a palavra começar por* `0`*, e o autómato consumisse* `0` *em* s_0 *e se mantivesse em* s_0*, então não memorizava o facto de esse* `0` *poder ser o início de* `001`*.*
- ...

O que memoriza cada estado *sobre o prefixo já lido?*

s_1**:** *palavras terminam em* `1` *mas não em* `001`*;*

s_3**:** *palavras terminam em* `10`*;*

s_4**:** *palavras terminam em* `00` *mas não em* `100`*;*

s_5**:** *palavra terminam em* `001`*;*

s_6**:** *palavra terminam em* `100`*;*

s_2**:** *palavra é* `0`*;* s_0*: a palavra é* ε*.*

3.1.4 Linguagem aceite por um autómato finito determinístico

Definição 3.2. *Dado um autómato finito* $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$*, uma palavra* $x \in \Sigma^\star$ *é* ***aceite*** *por* $\mathcal{A}$ *se* $\widehat{\delta}(s_0, x) \in F$*.*
A linguagem aceite por um autómato finito determinístico é definida por

$$L(\mathcal{A}) = \{x \in \Sigma^\star \mid \widehat{\delta}(s_0, x) \in F\}$$

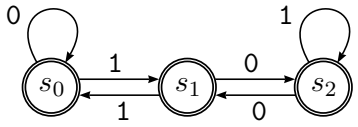
Se L *é* $L(\mathcal{A})$ *para algum AFD* $\mathcal{A}$*, dizemos que* L *é uma* ***linguagem regular****.*

Exemplo 3.8. Autómato para reconhecer múltiplos de 3 em binário
Pretende-se obter um autómato que reconheça palavras de $\{0,1\}^\star$ *que representem em binário inteiros múltiplos de 3, i.e., a linguagem:*

$$M3 = \{0, 11, 110, 1001, 1100, \ldots\}$$

Para não sobrecarregar a notação vamos denotar por x tanto o inteiro como a sua representação em binário. Dado um inteiro x, se x é múltiplo de 3, i.e. $\exists k\ x = 3k$, dizemos que x é $\dot{3}$. Senão ou x é $\dot{3}+1$ ($\exists k\ x = 3k+1$) ou $\dot{3}+2$ (i.e., $\exists k\ x = 3k+2$). Consideremos um autómato, com estados s_i, para $1 \leq i \leq 3$, tal que $x \in L(s_i)$ se e só se x é $\dot{3}+i$ ou seja $x \equiv i \bmod 3$, $i = 0, 1, 2$. Para determinar as transições, basta ver o que acontece para $x0$ ou $x1$, supondo que depois de lido x o autómato está no estado s_i, para $1 \leq i \leq 3$. Tem-se que $(x0) = 2x + 0$ e $(x1) = 2x + 1$. Então, basta considerar os 3 casos para x (e $\varepsilon \equiv 0$). Se x é $\dot{3}$ temos que $x0$ é $\dot{3}$ e $x1$ é $\dot{3}+1$. Se x é $\dot{3}+1$ temos que $x0$ é $\dot{3}+2$ e $x1$ é $\dot{3}$. Se Se x é $\dot{3}+2$ temos que $x0$ é $\dot{3}+1$ e $x1$ é $\dot{3}+2$. Temos então definido o autómato, $\mathcal{A} = (\{s_0, s_1, s_2\}, \{0,1\}, \delta, s_0, \{s_0\})$, com $\delta(s_i, c) = s_j$ com $j = (2i + c) \bmod 3$ e $c \in \{0,1\}$, ou

δ	0	1
$\rightarrow \star s_0$	s_0	s_1
s_1	s_2	s_0
s_2	s_1	s_2



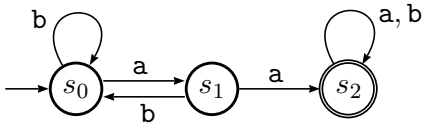
E facilmente se mostra que $\widehat{\delta}(s_0, x) = s_i$ sse $x \equiv i \bmod 3$ e que

$$M3 = L(\mathcal{A}) = \{x \in \Sigma^\star \mid \widehat{\delta}(s_0, x) = s_0\}$$

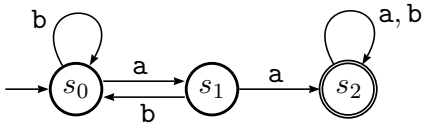
(Verificar!)

3.1.5 Complementar duma linguagem regular

Já vimos que a linguagem $L = \{x \mid x \in \{\mathtt{a}, \mathtt{b}\}^\star$ e tem `aa` como subpalavra$\}$ é a reconhecida pelo autómato



E que, a linguagem complementar
$\overline{L} = \{x \mid x \in \{\mathtt{a}, \mathtt{b}\}^\star$ e não tem `aa` como subpalavra$\}$ é reconhecida pelo autómato:



Se um autómato é completo (isto é, não encrava), podemos facilmente determinar um autómato que reconheça a linguagem complementar e temos então a seguinte proposição:

Proposição 3.1. *Se L é regular então $\overline{L} = \Sigma^\star \setminus L$ é regular.*

Prova. Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ o AFD (que não encrava) tal que $L = L(\mathcal{A})$.

Então $\overline{L} = L(\mathcal{B})$ onde $\mathcal{B} = (S, \Sigma, \delta, s_0, S \setminus F)$. Na realidade temos as seguintes equivalências:

$$x \in \overline{L} = L(\mathcal{B}) \Longleftrightarrow \widehat{\delta}(s_0, x) \in S \setminus F \Longleftrightarrow \widehat{\delta}(s_0, x) \notin F \Longleftrightarrow x \notin L(\mathcal{A}) = L$$

□

3.1.6 Exercícios de revisão

Exercício 3.1. *Descreva um autómato finito que reconheça a linguagem das palavras de $\{0,1\}^*$ que*

a) não têm nenhum `1`;

b) contêm pelo menos algum `0` e algum`1`;

c) têm comprimento não inferior a 2;

d) não contêm `101` como sub-palavra;

e) têm um número ímpar de `0` ou um número par de `1`;

f) têm no máximo um par de `0` e um par de `1` consecutivos;

g) não terminam em `100` nem em `001`

h) contêm (algures) pelo menos três "0" seguidos, mas não contêm dois ou mais "1" seguidos'

i) se têm algum par de "0" adjacentes, este aparece antes de qualquer par de "1" adjacentes;

j) não terminam em "1101" nem em "1011";

◇

3.2 Autómatos finitos não-determinísticos (AFNDs)

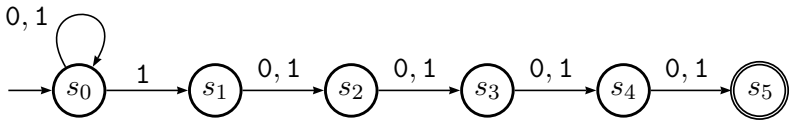
Uma computação diz-se **não-determinística** se o estado seguinte não for univocamente determinado pelo estado actual. Um autómato finito é não-determinístico se

- dado um estado s e um símbolo a existe um conjunto de estados possíveis seguintes, do qual se pode escolher um novo estado
- uma palavra é aceite, se começando no estado inicial existe um conjunto de escolhas (caminho) tal que quando acabar de a ler está num estado final.
- uma palavra é rejeitada se não existe nenhum caminho que leve dum estado inicial a um estado final.

O não-determinismo pode ser interpretado como a capacidade de escolher, de adivinhar, ou, ainda, de verificar. Seja a linguagem:

$$L = \{x \in \{0,1\}^\star \mid \text{o quinto símbolo a contar da direita é } 1\}$$

Um autómato finito não-determinístico que a reconhece poderá ser o seguinte:



Neste caso, para cada símbolo 1 lido o autómato pode escolher que ele é o quinto símbolo a contar da direita (muda para o estado s_1) ou continuar no estado s_0. Se $x \in L$ existe uma escolha que leva à aceitação; se $x \notin L$ nenhuma escolha leva à aceitação.

Exercício 3.2. *Para* 1010010 *determinar todos os possíveis caminhos no autómato dado e verificar se a palavra pertence a* L. $\diamond$

Notar que podemos construir um autómato finito determinístico que aceita L, mas tem pelo menos $2^5 = 32$ estados , pois tem de "recordar" os últimos 5 símbolos lidos! É imediato que um autómato finito determinístico é um caso particular dum autómato finito não determinístico. Mas o contrário, também se verifica. As linguagens aceites por autómatos finitos não determinísticos são precisamente as mesmas das aceites por autómatos finitos determinísticos. Isto é, para cada autómato finito não determinístico, existe um autómato finito determinístico que aceita a mesma linguagem. Informalmente, a ideia é associar um estado do AFD a cada conjunto de estados em que o AFND pode estar em cada passo da análise duma uma palavra. Temos que:

Estados do AFD subconjuntos do conjunto de estados do AFND

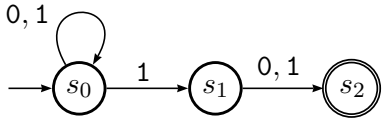
Estado inicial do AFD conjunto com o estado inicial do AFND

Estado final do AFD qualquer subconjunto de estados que contenha um estado final de AFND

Exemplo 3.9. *Considere-se a linguagem:*

$$L = \{x \in \{0,1\}^\star \mid \textit{o segundo símbolo a contar da direita é } 1\}$$

Um autómato não determinístico que a reconhece é:



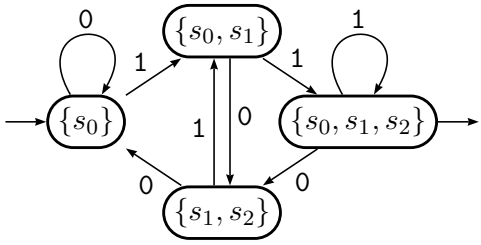
Por exemplo, após ler 001 *o autómato pode estar em* $\{s_0, s_1\}$*. Para construir um AFD equivalente determina-se para cada subconjunto de estados do AFND e símbolo do alfabeto, quais os estados que se obtém por transições de algum estado desse subconjunto. Por exemplo, se o subconjunto for* $\{s_0\}$*, por transições por* 0 *obtém* $\{s_0\}$*, mas por transições por* 1 *obtém-se os estados* $\{s_0, s_1\}$*. Se o subconjunto for* $\{s_0, s_1\}$*m por transições por* 0*, obtém-se o conjunto* $\{s_0, s_2\}$*, e por transições por* 1*, obtém-se o conjunto* $\{s_0, s_1, s_2\}$*, etc. Continuando este processo, temos que a função de transição* δ_d *do AFD equivalente é definido por:*

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

Podemos ver que começando em $\{s_0\}$ *por transições de* 0 *e* 1 *os estados* $\{s_1, s_2\}$*,* $\{s_1\}$*,* $\{s_2\}$ *e* $\emptyset$ *nunca são atingidos. Assim podemos eliminá-los e simplificar o AFD para:*

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

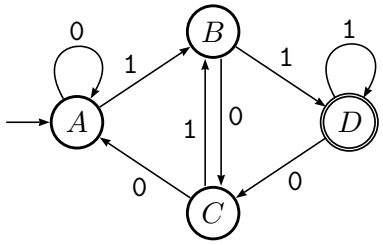
a que corresponde o diagrama seguinte:



No diagrama anterior o estado final é indicado com uma seta a sair do estado. Normalmente, renomeam-se os estados do autómato determinístico, por exemplo, para:

δ_d	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow A$	A	B
B	C	D
$\star C$	A	B
$\star D$	C	D

O ADF resultante corresponde precisamente a um que recorda *os dois últimos símbolos lidos: o estado $A = \{s_0\}$ (recorda 00), $B = \{s_0, s_1\}$ (01), $C = \{s_1, s_2\}$ (10) e $D = \{s_0, s_1, s_2\}$ (11).*



Definição 3.3. *Um autómato finito não determinístico $\mathcal{N}$ é um quinteto $(S, \Sigma, \delta, s_0, F)$ onde*

- *S o conjunto* finito *de* **estados**
- *Σ o* **alfabeto** *de símbolos de entrada*
- *$s_0 \in S$*
- *$F \subseteq S$ o conjunto de* **estados finais**
- *δ uma função de $S \times \Sigma$ em $\mathcal{P}(S)$,* ***função de transição***

Exemplo 3.10. *Para o Exemplo 3.9 a definição do autómato não determinístico é* $\mathcal{N} = (\{s_0, s_1, s_2\}, \{0,1\}, s_0, \delta, \{s_2\})$ *onde* δ *é dado por:*

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

3.2.1 Extensão da função de transição a palavras

Como no caso, dos AFD temos que estender a função de transição δ a palavras.
Seja $\widehat{\delta} : S \times \Sigma^\star \rightarrow \mathcal{P}(S)$ define-se $\widehat{\delta}(s,x)$ indutivamente por
comprimento de x, $|x|$:

Base. Se $x = \varepsilon$, $\widehat{\delta}(s, \varepsilon) = \{s\}$

Indução. Se $x = ya$ e $\widehat{\delta}(s,y) = \{p_1, \ldots p_k\}$, então $\widehat{\delta}(s, ya) = \cup_{i=1}^{k} \delta(p_i, a)$.

Isto é, para calcular $\widehat{\delta}(s, xa)$, calcula-se $\widehat{\delta}(s,x)$ e depois consideram-se todas as transições etiquetadas com a desses estados.

Exemplo 3.11. *Para processar* 0110 *no autómato do exemplo 3.10, temos:*

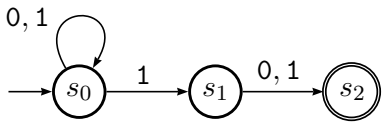
$$\begin{array}{lcllcllcl}
\widehat{\delta}(s_0, \varepsilon) & = & \{s_0\} & & & & & & \\
\widehat{\delta}(s_0, 0) & = & \{s_0\} & & & & & & \\
\widehat{\delta}(s_0, 01) & = & \delta(s_0, 1) & & = & \{s_0, s_1\} & & & \\
\widehat{\delta}(s_0, 011) & = & \delta(s_0, 1) \cup \delta(s_1, 1) & & = & \{s_0, s_1\} \cup \{s_2\} & = & \{s_0, s_1, s_2\} \\
\widehat{\delta}(s_0, 0110) & = & \delta(s_0, 0) \cup \delta(s_1, 0) \cup \delta(s_2, 0) & & = & \{s_0\} \cup \{s_2\} \cup \emptyset & = & \{s_0, s_2\}
\end{array}$$

3.2.2 Linguagem aceite por um AFND

Definição 3.4. *Seja* $\mathcal{N} = (S, \Sigma, \delta, s_0, F)$ *um AFND,* $\mathcal{N}$ *aceita* $x \in \Sigma^\star$ *se* $\widehat{\delta}(s_0, x) \cap F \neq \emptyset$.
A linguagem aceite por um autómato finito determinístico $\mathcal{N}$ *é definida por*

$$L(\mathcal{N}) = \{x \in \Sigma^\star \mid \mathcal{N} \text{ aceita } x, \text{ i.e, } \widehat{\delta}(s_0, x) \cap F \neq \emptyset\}$$

Exemplo 3.12. *Mostrar que* $L = \{x \in \{0,1\}^\star \mid$ *x termina em* 10 *ou* $11\} = L(\mathcal{N})$. *onde* $\mathcal{N}$ *é dado por:*



Vamos mostrar, por indução mutua em:

1. *Para todo o* x, $s_0 \in \widehat{\delta}(s_0, x)$
2. $s_1 \in \widehat{\delta}(s_0, x)$ *sse* x *termina em* 1
3. $s_2 \in \widehat{\delta}(s_0, x)$ *sse* x *termina em* 10 *ou* 11

Por indução no comprimento de x*:*

Base. $\mathbf{x} = \varepsilon$*, então* $\widehat{\delta}(s_0, \varepsilon) = \{s_0\}$*, logo (1) verifica-se. (2) e (3) também se verificam trivialmente.*

Indução. $\mathbf{x} = \mathbf{ya}$*, onde* a *é* 0 *ou* 1*, e supõe que (1-3) verificam-se para* $\widehat{\delta}(s_0, y)$*.*

(1) $s_0 \in \widehat{\delta}(s_0, y)$ *e como* $s_0 \in \delta(s_0, a)$*, então* $s_0 \in \widehat{\delta}(s_0, x)$

(2) ($\Rightarrow$) *Supor que* $a = 1$*, como* $s_0 \in \widehat{\delta}(s_0, y)$ *e* $s_1 \in \delta(s_0, 1)$*, então* $s_1 \in \widehat{\delta}(s_0, x)$.
($\Leftarrow$) *Supor que* $s_1 \in \widehat{\delta}(s_0, x)$*. Pela função de transição 3.10, só se pode chegar a* s_1 *se último símbolo for* 1*, i.e.,* $x = y1$.

(3) ($\Rightarrow$) *Se o penúltimo símbolo é* 1*, então* y *termina em* 1 *e* $s_1 \in \widehat{\delta}(s_0, x)$*. Mas* $s_2 \in \delta(s_1, a)$*, logo* $s_2 \in \widehat{\delta}(s_0, x)$.
($\Leftarrow$) *Supõe que* $s_2 \in \widehat{\delta}(s_0, x)$*. Pela tabela, só se pode chegar a* s_2 *de* s_1 *e para tal penúltimo símbolo de* x *tem de ser* 1.

3.2.3 Equivalência entre AFDs e AFNDs

Proposição 3.2. *Qualquer linguagem que seja reconhecida por um autómato finito determinístico pode ser reconhecida por algum autómato finito não determinístico.*

Prova. Um AFD $\mathcal{A} = (S, \Sigma, \delta_A, s_0, F)$ é equivalente a um AFND $\mathcal{N} = (S, \Sigma, \delta_N, s_0, F)$ com $\delta_N(s, a) = \{\delta_A(s, a)\}, \forall a \in \Sigma, s \in S$. Por indução em $|x|$ prova-se que $\widehat{\delta_A}(s_0, x) = s$ sse $\widehat{\delta_N}(s_0, x) = \{s\}$ □

Exercício 3.3. *Termina a demonstração anterior.* ◇

Proposição 3.3. *Qualquer linguagem que seja reconhecida por um autómato finito não determinístico pode ser reconhecida por algum autómato finito determinístico.*

Para provar a Proposição 3.3 temos que formalizar a construção feita no Exemplo 3.9, que se designa, habitualmente *construção por subconjuntos.*
Seja $\mathcal{N} = (S, \Sigma, \delta_N, s_0, F_N)$ um AFND. Construímos $\mathcal{A} = (\mathcal{P}(S), \Sigma, \delta_\mathcal{A}, \{s_0\}, F_\mathcal{A})$ onde

$$\begin{aligned} \delta_\mathcal{A}(A, a) &\stackrel{def}{=} = \cup_{s \in A} \delta_N(s, a)) \\ F_\mathcal{A} &\stackrel{def}{=} \{A \in \mathcal{P}(S) \mid A \cap F_N \neq \emptyset\} \end{aligned}$$

Lema 3.1. $\widehat{\delta}_{\mathcal{A}}(\{s_0\}, x) = \widehat{\delta}_N(s_0, x)$

Prova. Por indução em $|x|$.

Base. $x = \varepsilon$, por definição, $\widehat{\delta}_N(s_0, \varepsilon) = \{s_0\}$ e $\widehat{\delta}_A(\{s_0\}, \varepsilon) = \{s_0\}$

Indução. $x = ya$, onde a é o último símbolo de x. Por hipótese de indução, $\widehat{\delta}_{\mathcal{A}}(\{s_0\}, y) = \widehat{\delta}_N(s_0, y)$. Sejam estes estados $\{p_1, \ldots, p_k\}$. Por definição, $\widehat{\delta}_N(s_0, x) = \cup_{i=1}^{k} \delta_N(p_i, a)$ e $\delta_A(\{p_1, \ldots, p_k\}, a) = \cup_{i=1}^{k} \delta_N(p_i, a)$. Então,
$\widehat{\delta}_A(\{s_0\}, x) = \delta_A(\widehat{\delta}_A(\{s_0\}, y), a) = \delta_A(\{p_1, \ldots, p_k\}, a) = \cup_{i=1}^{k} \delta_N(p_i, a) = \widehat{\delta}_N(s_0, x)$

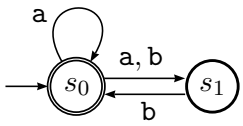
□

Prova. [**Proposição 3.3**] Para $x \in \Sigma^\star$,

$$\begin{aligned} x \in L(\mathcal{A}) &\iff \widehat{\delta}_A(\{s_0\}, x) \in F_{\mathcal{A}} \\ &\iff \widehat{\delta}_A(\{s_0\}, x) \cap F_N \neq \emptyset \\ &\iff \widehat{\delta}_N(s_0, x) \cap F_N \neq \emptyset \\ &\iff x \in L(\mathcal{N}) \end{aligned}$$

□

Exercício 3.4. *Considera o autómato finito não-determinístico representado pelo seguinte diagrama:*



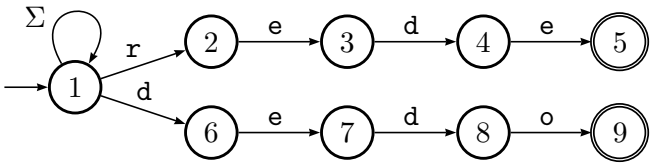
Converte, pela construção dos subconjuntos, o autómato num autómato finito determinístico.
◇

Nota: No pior caso, se o AFND tiver n estados, o menor AFD tem 2^n estados. Mas em muitos casos o AFD e o AFND têm essencialmente o mesmo número de estados, embora o AFD tenha muitas mais transições.

3.2.4 Aplicação: procura de um conjunto de palavras num texto

Suponhamos que temos um conjunto de palavras-chave que pretendemos saber se ocorrem num texto. Para tal podemos fazer um AFND que lê o texto e que entra num estado final se uma das palavras for encontrada.

Exemplo 3.13. *Seja Σ o conjunto das letras (minúsculas) do alfabeto do Português. O AFND seguinte reconhece a linguagem* $\mathtt{L} = \{\mathtt{rede}, \mathtt{dedo}\}$



onde , Σ *indica que para qualquer letra existe uma tranição do estado* 1 *para o estado* 1.
Agora podemos converter este autómato num AFD usando a construção de subconjuntos.

δ	r	e	d	o	$\Sigma \setminus \{r,e,d,o\}$
$\rightarrow \{1\}$	$\{1,2\}$	$\{1\}$	$\{1,6\}$	$\{1\}$	$\{1\}$
$\{1,2\}$	$\{1,2\}$	$\{1,3\}$	$\{1,6\}$	$\{1\}$	$\{1\}$
$\{1,6\}$	$\{1,2\}$	$\{1,7\}$	$\{1,6\}$	$\{1\}$	$\{1\}$
$\{1,3\}$	$\{1,2\}$	$\{1\}$	$\{1,4,6\}$	$\{1\}$	$\{1\}$
$\{1,7\}$	$\{1,2\}$	$\{1\}$	$\{1,6,8\}$	$\{1\}$	$\{1\}$
$\{1,4,6\}$	$\{1,2\}$	$\{1,5,7\}$	$\{1,6\}$	$\{1\}$	$\{1\}$
$\{1,6,8\}$	$\{1,2\}$	$\{1,7\}$	$\{1,6\}$	$\{1,9\}$	$\{1\}$
$\star\{1,5,7\}$	$\{1,2\}$	$\{1\}$	$\{1,6,8\}$	$\{1\}$	$\{1\}$
$\star\{1,9\}$	$\{1,2\}$	$\{1\}$	$\{1,6\}$	$\{1\}$	$\{1\}$

Mas para este tipo de AFND, que representam uma linguagem finita, podemos construir sempre um que o AFD equivalente que tenha no máximo o mesmo número de estados:

1. Se s_0 é o estado inicial do AFND, então $\{s_0\}$ é um dos estados do AFD
2. Seja s é um estado do AFND acessível de s_0 por um caminho cujos símbolos são $a_1a_2...a_m$. O AFD contém um estado composto por: s_0, s e todos os estados acessíveis de s_0 por um caminho etiquetado com sufixos de $a_1a_2...a_m$.

As transições podem ser obtidas pelo método da construção de subconjuntos, ou simplesmente: para cada estado do AFD $S = \{s_0, t_1, \ldots, t_k\}$, determinar para cada símbolo a

1. se existem transições de t_i etiquetadas por a, então S tem uma transição por a para o estado constituído por s_0 e todos os estados destino de transições de t_i por a
2. caso contrário, S tem uma transição por a para o estado constituído por s_0 e todos os estado destino de transições de s_0 por a.

Exercício 3.5. *a) Usando o método anterior, descreva a construção de AFND que reconhece um conjunto genérico de palavras.*

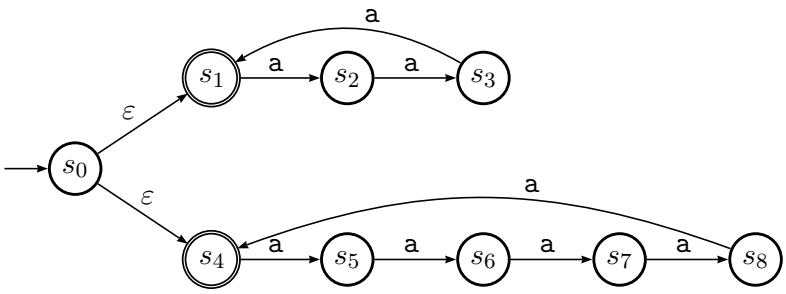
b) Construa um AFD equivalente ao dado no Exemplo 3.13.

c) Escreva um AFND, e um AFD correspondente, para reconhecer $\mathtt{ac}, \mathtt{bc}, \mathtt{dbc}$ *em* $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$.
◇

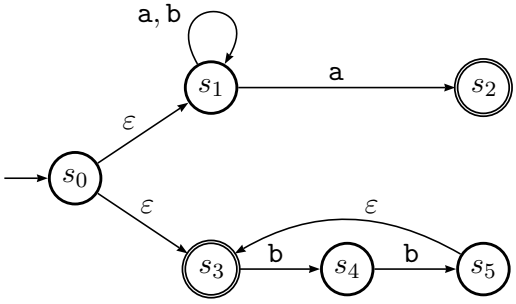
3.3 Autómatos finitos não determinísticos com transições por ε (AFND-ε)

Vamos agora considerar autómatos finitos que podem mudar de estado sem consumir qualquer símbolo, isto é, são autómatos finitos não determinísticos que admitem transições por ε (i.e., a palavra vazia). Como no caso dos autómatos não deterministicos estes automatos aceitam as mesmas linguagens que os autómatos determinísticos, e têm interesse especialmente teórico e em permitir a construção de auómatos mais compactos.

Exemplo 3.14. *Estes autómatos tornam trivial a construção de um autómato que reconhece a reunião de duas linguagens descritas por autómatos finitos. Basta considerar um novo estado inicial donde saem transições ε para os estados iniciais de cada um dos outros dois autómatos.*
O autómato seguinte aceita a linguagem $\{x \in \{a\}^\star \mid |x| \text{ é divisível por } 3 \text{ ou } 5\}$:



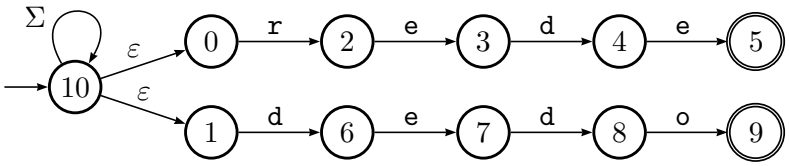
Exemplo 3.15. *Considere o seguinte autómato:*



O conjunto de estados possível após consumir aa *é $\{s_1, s_2, s_3\}$, pelo que* aa *é aceite; depois de consumir* bb *é $\{s_1, s_3, s_5\}$, pelo que* bb *é aceite; depois de consumir* abbab *é $\{s_1, s_4\}$, pelo que* abbab *não é aceite; depois de consumir* abbaba *é $\{s_1, s_2\}$, pelo que* abbaba *é aceite.*
Verifique que a linguagem aceite pelo autómato é

$$\{x \in \{\mathtt{a}, \mathtt{b}\}^\star \mid x \text{ termina em } \mathtt{a}\} \cup \{\mathtt{bb}, \mathtt{a}\}^\star$$

Exemplo 3.16. *Podemos também simplificar a construção dos autómatos que reconhecem conjuntos finitos de palavras-chave. Para a linguagem do exemplo 3.13, podiamos ter o autómato:*



Construímos um autómato para cada palavra (cada uma com o seu estado inicial) e depois reunimos todos num só com um novo estado inicial e transições por ε desse estado para cada um dos iniciais das palavras.

Definição 3.5. *Um autómato finito não determinístico $\mathcal{E}$ com transições por ε é um quinteto $(S, \Sigma, \delta, s_0, F)$, como para os AFNDs mas onde em que δ é uma função de $S \times (\Sigma \cup \{\varepsilon\})$ em $\mathcal{P}(S)$*

Exemplo 3.17. *Vamos construir um AFND-ε que reconhece a linguagem das palavras que representam números decimais, i.e, palavras que:*

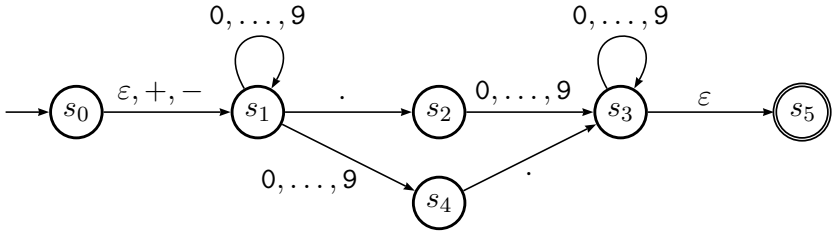
- *opcionamente, contém um sinal $+$ ou $-$*
- *uma sequência de digitos 0 a 9*
- *um ponto decimal $\cdot$*
- *outra sequência de dígitos*
- *e pelo menos uma das sequência de dígitos tem de ser não vazia*

Seja

$$\mathcal{E} = (\{s_0, s_1, \ldots, s_5\}, \{\cdot, +, -, 0, 1, \ldots, 9\}, \delta, s_0, \{s_5\})$$

onde δ é dado por:

	ε	$-, +$	$\cdot$	$0, 1, \ldots 9$
s_0	$\{s_1\}$	$\{s_1\}$	$\emptyset$	$\emptyset$
s_1	$\emptyset$	$\emptyset$	$\{s_2\}$	$\{s_1, s_4\}$
s_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{s_3\}$
s_3	$\{s_5\}$	$\emptyset$	$\emptyset$	$\{s_3\}$
s_4	$\emptyset$	$\emptyset$	$\{s_3\}$	$\emptyset$
s_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



3.3.1 Fecho por transições ε

Dado um AFDN-ε, $E_\varepsilon = (S, \Sigma \cup \{\varepsilon\}, \delta, s_0, F)$, para cada estado $s \in S$, podemos definir o conjunto de estados acessíveis do estado s por transições por ε, que se denota por $Fecho_\varepsilon(s)$. Formalmente definimos $Fecho_\varepsilon(s)$ recursivamente por

Base. $s \in Fecho_\varepsilon(s)$

Indução. se $p \in Fecho_\varepsilon(s)$ e $r \in \delta(p, \varepsilon)$ então $r \in Fecho_\varepsilon(s)$

Exemplo 3.18. *Para o autómato do Exemplo 3.18 temos:*

$$\begin{aligned} Fecho_\varepsilon(s_0) &= \{s_0, s_1\} \\ Fecho_\varepsilon(s_3) &= \{s_3, s_5\} \\ Fecho_\varepsilon(s) &= \{s\} \textit{para os restantes estados } s \end{aligned}$$

3.3.2 Extensão da função transição a palavras

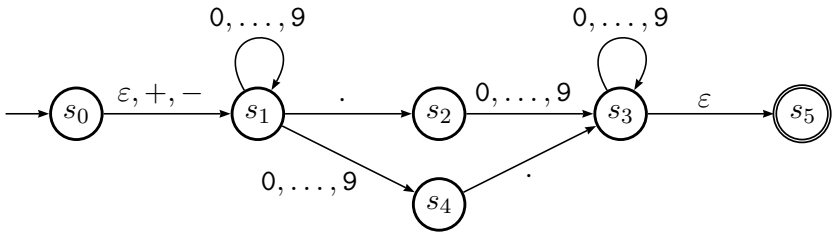
Dado $E_\varepsilon = (S, \Sigma \cup \{\varepsilon\}, \delta, s_0, F)$ estendemos $\widehat{\delta}$ de modo a que $\widehat{\delta}(s, w)$ seja o conjunto de estados que são acessíveis de s por caminhos cujas etiquetas concatenadas dão w, mas algumas podem ser ε. A definição recursiva é:

Base. $\widehat{\delta}(s, \varepsilon) = Fecho_\varepsilon(s)$

Indução. Suponhamos que $w = xa$, com $a \in \Sigma$,

- seja $\widehat{\delta}(s, x) = \{p_1, \ldots, p_k\}$
- e seja $\cup_{i=1}^{k} \delta(p_i, a) = \{r_1, \ldots, r_m\}$
- então $\widehat{\delta}(s, w) = \cup_{j=1}^{m} Fecho_\varepsilon(r_j)$

Exemplo 3.19. *Para o autómato do Exemplo 3.18,*



calculemos $\widehat{\delta}(s_0, 3.7)$*:*

- $\widehat{\delta}(s_0, \varepsilon) = Fecho_\varepsilon(s_0) = \{s_0, s_1\}$
- *Para* $\widehat{\delta}(s_0, 3)$
  - $\delta(s_0, 3) \cup \delta(s_1, 3) = \emptyset \cup \{s_1, s_4\}$
  - $\widehat{\delta}(s_0, 3) = Fecho_\varepsilon(s_1) \cup Fecho_\varepsilon(s_4) = \{s_1, s_4\}$
- *Para* $\widehat{\delta}(s_0, 3.)$
  - $\delta(s_1, \cdot) \cup \delta(s_4, \cdot) = \{s_2\} \cup \{s_3\} = \{s_2, s_3\}$
  - $\widehat{\delta}(s_0, 3.) = Fecho_\varepsilon(s_2) \cup Fecho_\varepsilon(s_3) = \{s_2, s_3, s_5\}$
- *Para* $\widehat{\delta}(s_0, 3.7)$
  - $\delta(s_2, 7) \cup \delta(s_3, 7) \cup \delta(s_5, 7) = \{s_3\}$
  - $\widehat{\delta}(s_0, 3.7) = Fecho_\varepsilon(s_3) = \{s_3, s_5\}$

A linguagem aceite por um AFND-ε define-se então como para os AFNDs:

Definição 3.6. *Dado um automato não determinístico com transições por* ε*,* $E_\varepsilon = (S, \Sigma \cup \{\varepsilon\}, \delta, s_0, F)$ *a linguagem a aceite por* E *é definida por*

$$L(E) = \{x \in \Sigma^\star \mid \widehat{\delta}(s_0, x) \cap F \neq \emptyset\}$$

Para o Exemplo 3.18 como $\widehat{\delta}(s_0, 3.7) = \{s_3, s_5\}$ e $s_5 \in F$ então $3.7 \in L(E)$.

3.3.3 Eliminação de transições ε

Dado $\mathcal{E} = (S_E, \Sigma, \delta_E, s_0, F_E)$ um AFND-ε, podemos construímos um AFD $A = (S_A, \Sigma, \delta_A, s_A, F_A)$ equivalente, por um método semelhante à da construção de subconjuntos, tal que:

- $s_A = Fecho_\varepsilon(s_0)$
- S_A é o conjunto dos subconjuntos X de S_E, tal que $X = Fecho_\varepsilon(X)$ (i.e X é fechado por transições ε) e X é acessível de s_A
- $F_A = \{X \mid X \in S_A \text{ e } X \cap F_E \neq \emptyset\}$
- $\delta_A(X, a)$ é calculado para $a \in \Sigma$ e $X \in S_A$ por
  1. Seja $X = \{p_1, \ldots, p_k\}$

2. Seja $\cup_{i=1}^{k}\delta(p_i,a)=\{r_1,\ldots,r_m\}$
3. então $\delta_A(X,a)=\cup_{j=1}^{m}Fecho_\varepsilon(r_j)$

Exemplo 3.20. *Vamos construir um ADF equivalente ao AFND-ε do exemplo 3.18. Começamos por determinar o Fecho-ε de cada estado de $\mathcal{E}$:*

$$\begin{aligned} Fecho_\varepsilon(s_0) &= \{s_0,s_1\} \\ Fecho_\varepsilon(s_3) &= \{s_3,s_5\} \\ Fecho_\varepsilon(s) &= \{s\} \quad \textit{e para os restantes estados } s \end{aligned}$$

Pela construção anterior obtém-se:

	$-,+$	$\cdot$	$0,1,\ldots 9$
$\rightarrow \{s_0,s_1\}$	$\{s_1\}$	$\{s_2\}$	$\{s_1,s_4\}$
$\{s_1\}$	$\emptyset$	$\{s_2\}$	$\{s_1,s_4\}$
$\{s_2\}$	$\emptyset$	$\emptyset$	$\{s_3,s_5\}$
$\{s_1,s_4\}$	$\emptyset$	$\{s_3,s_2,s_5\}$	$\{s_1,s_4\}$
$\star\{s_3,s_5\}$	$\emptyset$	$\emptyset$	$\{s_3,s_5\}$
$\star\{s_2,s_3,s_5\}$	$\emptyset$	$\emptyset$	$\{s_3,s_5\}$

3.3.4 Equivalência entre AFD e AFND-ε

Proposição 3.4. *Uma linguagem L sobre um alfabeto Σ é aceite por um AFND-ε se e só se L é aceite por algum AFD.*

Lema 3.2. *Para todo $x\in\Sigma^\star$, $\widehat{\delta}_E(s_0,x)=\widehat{\delta}_A(s_A,x)$*

Prova. Por indução em $|x|$.

Base. Se $x=\varepsilon$, $\widehat{\delta}_E(\{s_0\},\varepsilon)=Fecho_\varepsilon(s_0)$ e $s_A=Fecho_\varepsilon(s_0)$. Então $\widehat{\delta}_A(s_A,\varepsilon)=s_A=Fecho_\varepsilon(s_0)=\widehat{\delta}_E(s_0,\varepsilon)$

Indução. Suponha que $x=ya$, para algum $a\in\Sigma$, e $\widehat{\delta}_E(s_0,y)=\widehat{\delta}_A(s_A,y)$, sejam $\{p_1,\ldots,p_k\}$. Pela definição calculamos $\widehat{\delta}_E(s_0,x)$ por

1. $\cup_{i=1}^{k}\delta(p_i,a)=\{r_1,\ldots,r_m\}$
2. $\widehat{\delta}_E(s_0,x)=\cup_{j=1}^{m}Fecho_\varepsilon(r_j)$

mas isso é precisamente $\delta_A(\{p_1,\ldots,p_k\},a)$ que é $\widehat{\delta}_A(s_A,x)$.

□

Prova. (Proposição 3.4)

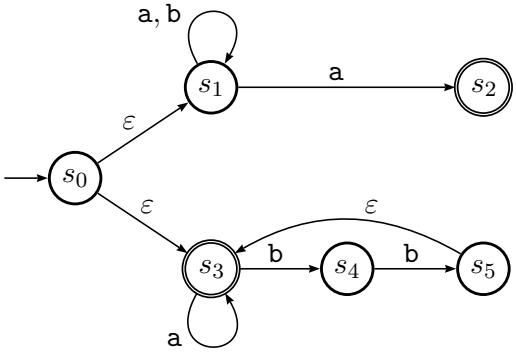
($\Leftarrow$) Seja $\mathcal{E} = (S_E, \Sigma, \delta_E, s_0, F_E)$ um AFND-ε e seja $\mathcal{A} = (S_A, \Sigma, \delta_A, s_A, F_A)$ um AFD construído pelo método de subconjuntos modificado. Queremos que $L(\mathcal{A}) = L(\mathcal{E})$.

Para $x \in \Sigma^\star$,

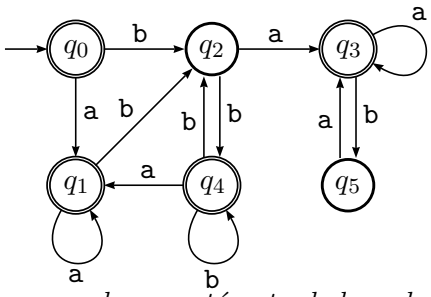
$$\begin{aligned} x \in L(\mathcal{A}) &\iff \widehat{\delta}_A(s_A, x) \in F_{\mathcal{A}} \\ &\iff \widehat{\delta}_A(s_A, x) \cap F_E \neq \emptyset \\ &\iff \widehat{\delta}_E(s_0, x) \cap F_E \neq \emptyset \\ &\iff x \in L(\mathcal{E}) \end{aligned}$$

($\Rightarrow$) Para tornar um AFD num AFND-ε basta acrescentar $\delta(s, \varepsilon) = \emptyset$ e transformar as transições do tipo $\delta(s, a) = s'$ em $\delta(s, a) = \{s'\}$, para todos os estados do AFD. □

Exemplo 3.21. *Para o autómato do Exemplo 3.15*



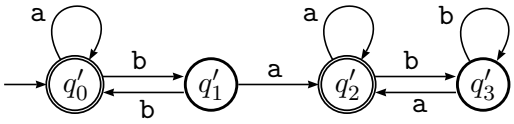
é equivalente ao autómato finito determinístico seguinte,



cujo estado inicial q_0 corresponde ao autómato dado poder estar inicialmente, sem que tenha consumido qualquer símbolo, em $\{s_0, s_1, s_3\}$.

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

o qual não é o **AFD minímo** *(com menor número de estados e que não encrava). O AFD minímo equivalente aos anteriores é o seguinte:*



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, tente convencer-se, neste caso, da necessidade de cada uma das transições efectuadas. Para isso note que se a palavra dada não pertencer a* $\{\mathtt{bb}, \mathtt{a}\}^\star$ *tem algum* `a` *imediatamente após um bloco maximal de* `b`*'s em número ímpar e nesse caso terá que terminar em* `a`*.*

Exercício 3.6. *Considera o autómato finito com transições por* ε*,* $(\{s_0, s_1, s_2\}, \{a, b, c\}, \delta, s_0, \{s_2\})$ *com a seguinte função de transição* δ*:*

	ε	a	b	c
s_0	$\{s_1, s_2\}$	$\emptyset$	$\{s_1\}$	$\{s_2\}$
s_1	$\emptyset$	$\{s_0\}$	$\{s_2\}$	$\{s_0, s_1\}$
s_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

a) Apresente o diagrama que descreve o autómato.

b) Calcule o fecho-ε *de cada estado.*

c) Determine todas as palavras com comprimento ≤ 3 *aceites pelo autómato.*

d) Determine um autómato finito determinístico completo equivalente.

$\diamond$

Capítulo 4

Expressões Regulares e Autómatos Finitos

4.1 Expressões Regulares

Embora os autómatos finitos permitam caracterizar de um modo finito uma dada linguagem, muitas vezes é mais conveniente ter uma que descrição declarativa dos padrões das palavras que constituem uma dada linguagem. No caso das linguagens aceites por autómatos finitos vamos ver que elas podem ser descritas por **expressões regulares**. Variantes destas expressões são usadas em várias aplicações computacionais como

- expansão de nomes de ficheiros em `UNIX`: `ls *.c`
- em procura de palavras em comandos como o `grep` ou navegadores `WWW`
- analisadores lexicais (`lex`) de compiladores ou processadores de linguagens naturais: ex. números decimais, identificadores, palavras chave, etc.

Exemplo 4.1. *A linguagem*

$$L_1 = \{x \in \{0,1\}^\star \mid x \text{ tem 2 ou 3 ocorrências de } 1, \text{ não sendo as duas primeiras consecutivas } \}$$

pode ser descrita por:
$L_1 = \{0\}^\star\{1\}\{0\}^\star\{0\}\{1\}\{0\}^\star(\{1\}\{0\}^\star \cup \{\varepsilon\})$
ou simplificando, podemos representar a linguagem L_1 pela expressão

$$0^\star 10^\star 010^\star(10^\star + \varepsilon)$$

Exemplo 4.2. *A linguagem*

$$L_2 \quad = \quad \{x \in \{0,1\}^\star \mid x \text{ contém } 000\}$$

pode ser descrita por:
$L_2 = \{0,1\}^\star\{000\}\{0,1\}^\star$
ou simplificando, podemos associar a L_2 *a expressão:*

$$(0+1)^\star 000(0+1)^\star$$

Qualquer expressão regular r representa uma linguagem que se designa por $L(r)$.

Definição 4.1. *O conjunto das* **expressões regulares sobre um alfabeto** Σ *e o conjunto das linguagens por elas descritas são definidos indutivamente por:*

1. ε *é uma expressão regular sobre* Σ*, e descreve a linguagem* $\{\varepsilon\}$*; i.e* $L(\varepsilon) = \{\varepsilon\}$
2. $\emptyset$ *é uma expressão regular sobre* Σ*, e descreve a linguagem* $\emptyset$*; i.e* $L(\emptyset) = \emptyset$
3. *Se* $a \in \Sigma$ *então* a *é uma expressão regular sobre* Σ*, e descreve a linguagem* $\{a\}$*; i.e* $L(a) = \{a\}$
4. *Se* r *e* s *são expressões regulares sobre* Σ *que descrevem as linguagens* $L(r)$ *e* $L(s)$*, então* $(r+s)$*,* (rs) *e* $(r^\star)$ *são expressões regulares sobre* Σ*, e descrevem* $L(r) \cup L(s)$*,* $L(r)L(s)$ *e* $L(r)^\star$ *respectivamente;i.e* $L((r+s)) = L(r) \cup L(s)$*,* $L((rs)) = L(r)L(s)$ *e* $L((r^\star)) = L(r)^\star$*.*
5. *As expressões regulares sobre* Σ *e as linguagens por elas descritas são todas e apenas as obtidas por 1-4.*

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

Exemplo 4.3. *Seja* $\Sigma = \{0,1\}$*.*

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

Pela definição de expressão regular, facilmente se concluí que o conjunto das linguagens regulares sobre um alfabeto Σ contém $\emptyset$, $\{\varepsilon\}$, $\{a\}$ para $a \in \Sigma$ e é fechado para a concatenação, reunião e fecho de Kleene. No entanto, iremos ver que nem toadas as linguagens são regulares. Por exemplo, a linguagem $\{0^n 1^n \mid n \geq 1\}$ não o é.
Usando uma convenção idêntica à aplicada a expressões aritméticas envolvendo soma, produto e potenciação, podemos abreviar as expressões retirando parêntesis "desnecessários", como ilustrado na Figura 4.1. As regras de precedência para as operações + (união), concatenação e * (fecho de Kleene) correspondem às usadas nas expressões aritméticas para a adição, a multiplicação e a potenciação, respectivamente. Isto é, a precedência é fecho de Kleene > concatenação > união. A vantagem da introdução desta simplificação é claramente ilustrada pelos exemplos dados na Figura 4.1.

$(r+s)$	$r+s$
$((r+s)+t)$	$r+s+t$
$(r(st))$	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 4.1: Expressão regular e abreviatura respectiva

Exemplo 4.4. *Mais alguns exemplos de linguagens de alfabeto* $\Sigma = \{\mathtt{0},\mathtt{1}\}$ *e de expressões regulares que as descrevem.*

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

Exemplo 4.5. *Para cada uma das linguagens seguintes vamos determinar uma expressão regular que a representa.*

- *Seja* $A = \{x \in \{0,1\}^\star \mid x$ *tem pelo menos um* 11 *entre cada par de* 0*'s*$\}$

  Numa palavra de A*, se ocorrer um* 0 *tem de ocorrer* 11 *a seguir. Temos então a expressão regular*

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

  Mas, excepto para o 0 *mais à direita. Então, temos que*

  $$A = L((1 + 011)^\star(\varepsilon + 0 + 01))$$

- *Seja* $B = \{x \in \{\mathtt{a}, \mathtt{b}\}^\star \mid x$ *que tem um número par de* a*'s*$\}$

  Se uma palavra de B *tiver um* a *tem necessariamente outro. Temos então que:*

  $$B = L((\mathtt{ab}^\star\mathtt{a} + \mathtt{b})^\star)$$

  Note que se uma palavra tiver número par de a*'s, e esse número não for zero, então a palavra pode ser vista como uma sequência de* b*'s e de palavras em* $L(\mathtt{ab}^\star\mathtt{a})$*.*

  Por exemplo, para babbbbbabbbbbbbbbaaaa *tem-se a seguinte decomposição*

  $$\underbrace{\mathtt{b}}_{\in L(\mathtt{b}^\star)}\ \underbrace{\mathtt{abbbbba}}_{\in L(\mathtt{ab}^\star\mathtt{a})}\ \underbrace{\mathtt{bbbbbbbbb}}_{\in L(\mathtt{b}^\star)}\ \underbrace{\mathtt{aa}}_{\in L(\mathtt{ab}^\star\mathtt{a})}\ \underbrace{\mathtt{aa}}_{\in L(\mathtt{ab}^\star\mathtt{a})}$$

- *Seja* $C = \{x \in \{0,1\}^\star \mid x$ *não tem a subpalavra* 111$\}$

  Numa palavra de C*, se houver um* 1 *ou um* 11 *tem de haver pelo menos um* 0*. Isto é,*

  $$C = L((0^\star + 0^\star(1 + 11)(0^+(1 + 11))^\star 0^\star))$$

- *Seja* $D = \{x \in \{0,1\}^\star \mid$ *x não tem a subpalavra* `00`$\}$

  Numa palavra de D, à direita de um `0` *tem de haver um* `1`*. Temos então a expressão* $(01+1)^\star$*. Mas a palavra pode terminar em* 0*. Fica, então,*

  $$D = L((01+1)^\star(0+\varepsilon))$$

- *Seja R, linguagem dos números decimais racionais em* $\Sigma = \{0,1,2,3,4,5,6,7,8,9,-,.\}$

  Para os dígitos temos a expressão regular: $digito = (0+1+2+3+4+5+6+7+8+9)$*. E,*

  $$R = L((\varepsilon + -)digito^+(.digito^+ + \varepsilon))$$

Exercício 4.1. *Melhorar a expressão obtida no exemplo anterior para a linguagem R, para que não existam* 0*'s à esquerda não significativos.* $\diamond$

4.1.1 Equivalência de expressões regulares

Seja R_Σ o conjunto das expressões regulares sobre Σ e seja $\equiv$ a relação binária em R_Σ assim definida:

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

quaisquer que sejam $r, s \in R_\Sigma$.
Pode verificar-se que $\equiv$ é uma relação de equivalência (reflexiva, simétrica e transitiva. Diz-se que duas expressões regulares são **equivalentes** se e só se as linguagens por elas descritas são iguais.

Exercício 4.2. *Mostrar que a relação* $\equiv$ *é de equivalência.* $\diamond$

Proposição 4.1. *Quaisquer que sejam as expressões regulares r, s e t de alfabeto* Σ *tem-se:*

$r+s$	$\equiv$	$s+r$			*(comutatividade +)*
$(r+s)+t$	$\equiv$	$r+(s+t)$			*(associatividade +)*
$r(st)$	$\equiv$	$(rs)t$			*(associatividade $\cdot$)*
$r(s+t)$	$\equiv$	$(rs)+(rt)$			*(distributividade + $\cdot$)*
$(r+s)t$	$\equiv$	$rt+st$			
$r+\emptyset$	$\equiv$	r			
$r+r$	$\equiv$	r			
εr	$\equiv$	r	$\equiv$	$r\varepsilon$	*(elemento neutro $\cdot$)*
$\emptyset r$	$\equiv$	$\emptyset$	$\equiv$	$r\emptyset$	*(elemento absorvente $\cdot$)*
$\emptyset^\star$	$\equiv$	$\varepsilon^\star$	$\equiv$	ε	*(propriedades do fecho)*
$(\varepsilon+r)^\star$	$\equiv$	$r^\star$			
$\varepsilon+rr^\star$	$\equiv$	$\varepsilon+r^\star r$	$\equiv$	$r^\star$	
$(r^\star)^\star$	$\equiv$	$r^\star$			
$(r^\star s^\star)^\star$	$\equiv$	$(r+s)^\star$			
$r(sr)^\star$	$\equiv$	$(rs)^\star r$			
$(r^\star s)^\star r^\star$	$\equiv$	$(r+s)^\star$			

Prova. Segue imediatamente como corolário da Proposição 2.1 usando a definição de expressão regular (e linguagem por ela descrita). □

Exemplo 4.6. *Analise ainda a expressão regular*

$$\mathtt{1}^\star + \mathtt{1}^\star\mathtt{0}(\mathtt{110})^\star\mathtt{1}^\star$$

a qual é equivalente à expressão

$$\mathtt{1}^\star(\varepsilon + \mathtt{0}(\mathtt{110})^\star\mathtt{1}^\star)$$

e define a linguagem das palavras que têm exactamente dois 1*'s entre cada duas ocorrências consecutivas de* 0*'s. Note que se x é uma palavra tal que entre cada duas ocorrências consecutivas de* 0*'s há exactamente dois* 1*'s, então se x for da forma*

$$x_1\mathtt{0}x_2$$

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

Exemplo 4.7. *Considere ainda mais alguns exemplos.*

- *A linguagem das palavras em* $\{\mathtt{a},\mathtt{b}\}^\star$ *que têm* $\mathtt{aa}$ *como prefixo e* $\mathtt{bbb}$ *como sufixo é*

$$\{\mathtt{aa}\}\{\mathtt{a},\mathtt{b}\}^\star\{\mathtt{bbb}\},$$

sendo descrita, por exemplo, pela expressão regular $\mathtt{aa}(\mathtt{a}+\mathtt{b})^\star\mathtt{bbb}$.

- *A linguagem* $\{\mathtt{ab}^n\mathtt{a} \mid n \in \mathbb{N}, n \geq 3\}$ *de alfabeto* $\{\mathtt{a},\mathtt{b}\}$ *é descrita pela expressão regular* $\mathtt{abbbb}^\star\mathtt{a}$.

- *A linguagem* $\{\mathtt{0b}^{3n}\mathtt{0}^{2m} \mid n, m \in \mathbb{N}\}$ *de alfabeto* $\{\mathtt{0},\mathtt{b}\}$ *é descrita pela expressão regular*

$$\mathtt{0}(\mathtt{bbb})^\star(\mathtt{00})^\star$$

Mais adiante, vamos ver que $\{\mathtt{0b}^{3n}\mathtt{0}^{2n} \mid n \in \mathbb{N}\}$ **não é uma linguagem regular**, *ou seja, não existe uma expressão regular que a descreve. Note que neste caso, o número de* $\mathtt{0}$*'s na palavra excede numa unidade dois terços do número de* $\mathtt{b}$*'s.*

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

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

- A linguagem das palavras de alfabeto $\{\mathtt{0},\mathtt{1}\}$ que têm $\mathtt{00}$ como subpalavra *é*

$$\{\mathtt{0},\mathtt{1}\}^\star\{\mathtt{00}\}\{\mathtt{0},\mathtt{1}\}^\star$$

a qual é descrita pela expressão regular

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

Uma expressão regular equivalente (ou seja, que descreve exactamente a mesma linguagem) é por exemplo,

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

Note que, a expressão regular

$$(\mathtt{01}+\mathtt{1})^\star$$

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

- A linguagem das palavras em $\{\mathtt{0},\mathtt{1}\}^\star$ que não têm $\mathtt{0}$'s consecutivos *é descrita pela expressão regular* $(\mathtt{01}+\mathtt{1})^\star(\mathtt{0}+\varepsilon)$.

4.1.2 Expressões regulares e padrões

Em várias aplicações e linguagens de programação, usam-se variantes das expressões regulares, denominados também por *padrões*, para descrever e reconhecer conjuntos de palavras (`strings`) que correspondem a linguagens regulares:

- em linguagens de programação que manipulam `strings`: `bash`, `tcl`, `perl`, `python`, etc
- reconhecedores de padrões em ficheiros, como o `grep`: `grep 'ex*' teste.txt`
- analisadores lexicais, como `lex`, que permitem seccionar um texto em elementos identificados (`tokens`): p.e num programa em `C`, identificar o que são palavras reservadas (*if, while,struct,...*), variáveis, constantes, operadores, etc. São usados pelos compiladores.
- editores de texto, como o `emacs`, na procura de palavras.

Um *padrão* é uma sequência de símbolos duma dada forma que representa uma linguagem num dado alfabeto. O conjunto de palavras que são reconhecidas por um dado padrão α denota-se por $L(\alpha)$.

4.1.2.1 Padrões para expansão de nomes de ficheiros (UNIX/bash)

O alfabeto é o conjunto de caracteres do código ASCII (ou UNICODE). Os padrões normalmente usados são os da tabela seguinte.

Padrão	Reconhece	Exemplo
`a`	o caracter `a`	`ls -l ola`
`?`	qualquer caracter do alfabeto	`ls -l ex?.c`
`*`	zero ou mais caracteres	`ls -l *.c`
`[...]`	qualquer caracter da lista	`ls -l ex*.[ch]`
`[c-d]`	qualquer caracter entre dois caracteres separados por `-`	`ls -l ex[1-9].c`
`[^...]`	caracteres que não pertencem à lista	`ls -l ex[^12].c`
`{...}`	conjuntos alternativos	`ls -l *.{gif,jpg}`

Exercício 4.3. *Verificar que são regulares as linguagens descritas por cada um dos padrões acima. Nota a diferença do uso do símbolo* `*`*. E o que significa* `?`*.* $\diamond$

4.1.2.2 Regexp (usadas no `grep`, em `tcl`, `perl`, ...)

Padrão atómico	Reconhece	Exemplo
`a`	o caracter `a`	`a`
`\a`	o caracter especial `a`	`\n`
`.`	qualquer caracter	`.`
`[...]`	qualquer caracter da lista	`[abc]`
	qualquer caracter entre dois caracteres separados por `-`	`[a-z]`
`[^...]`	caracteres que não pertencem à lista	`[^0-9]`
`^`	o inicio da palavra	`^a`
`$`	o fim da palavra	`a$`

Padrão composto	Reconhece	
$regex\|regex$	qualquer dos padrões	`[0-9]|[+-]`
$regexregex$	concatenação de padrões	`[0-9][a-z]`
$regex^\star$	zero ou mais $regex$	$[0-9]^\star$
$regex^+$	um ou mais $regex$	$[0-9]^+$
$regex?$	zero ou um $regex$	`[0-9]?`
$(regex)$	$regex$ (permite agrupamentos)	`(a|o)`

4.2 Equivalência entre Autómatos Finitos e Expressões Regulares

Vamos ver que as linguagens aceites por autómatos finitos são as descritas por expressões regulares, i.e.,:

- Qualquer linguagem descrita por uma expressão regular é aceite por um autómato finito
- Qualquer linguagem aceite por um autómato finito é descrita por uma expressão regular

4.2.1 De Expressões Regulares para Autómatos Finitos

Proposição 4.2. *Qualquer que seja a expressão regular r, existe um autómato finito que reconhece $L(r)$.*

Prova. (Método de Thompson)

Mostra-se por indução no número de operadores que ocorrem numa expressão regular r, que existe um AFND-ε N tal que $L(N) = L(r)$ e:

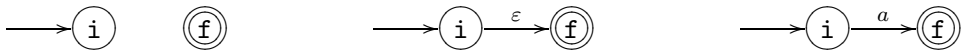
- N tem só um estado final
- Nenhuma transição para o estado inicial
- Nenhuma transição do estado final

Base. Se r tem 0 operadores, então ou:

1. r é $\emptyset$ e $L(\emptyset) = \emptyset$
2. r é ε e $L(\varepsilon) = \{\varepsilon\}$
3. $r = a$ para algum $a \in \Sigma$, e $L(a) = \{a\}$.

Os autómatos seguintes verificam as condições impostas:

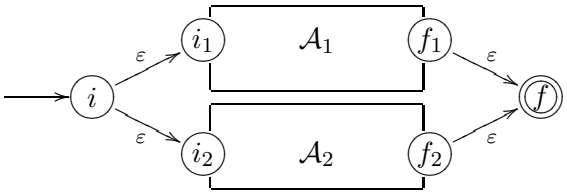
$r = \emptyset$ $\quad$ $r = \varepsilon$ $\quad$ $r = a$



Indução. Suponhamos que a proposição é verdadeira para expressões regulares com n ou menos operadores ($n \geq 1$). Se r tem $n+1$ operadores então:

Caso $r = (r_1 + r_2)$**.** Temos que $L(r_1 + r_2) = L(r_1) \cup L(r_2)$ e supondo que $L(r_i) = L(\mathcal{A}_i)$, $i = 1, 2$ e $\mathcal{A}_i = (S_i, \Sigma, \delta_i, i_i, \{f_i\})$ e, temos $L(\mathcal{A}_\cup) = L(r_1) \cup L(r_2)$ onde

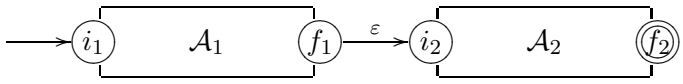
$$\mathcal{A}_\cup = (S_1 \cup S_2 \cup \{i, f\}, \Sigma, \delta, i, f) \text{ e}$$
$$\delta = \delta_1 \cup \delta_2 \cup \{(i, \varepsilon, i_1), (i, \varepsilon, i_2), (f_1, \varepsilon, f), (f_2, \varepsilon, f)\}$$



e o autómato $\mathcal{A}_\cup$ verifica as condições impostas.

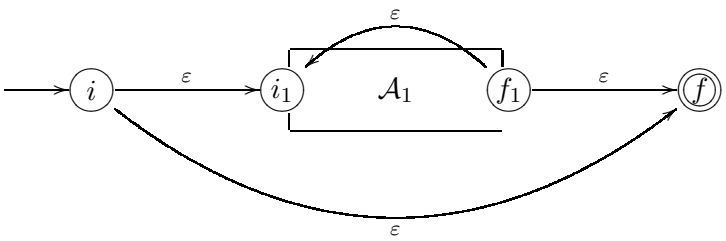
Caso $r = (r_1 r_2)$**.** . Temos que $L(r_1 r_2) = L(r_1)L(r_2)$ e supondo que $L(r_1) = L(\mathcal{A}_1)$ e $L(r_2) = L(\mathcal{A}_2)$, temos $L(\mathcal{A}_\cdot) = L(r_1)L(r_2)$

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



Caso $r = r_1{}^\star$**.** Temos que $L(r_1{}^\star) = L(r_1)^\star$ e supondo que $L(r_1) = L(\mathcal{A}_1)$ temos $L(r_1{}^\star) = L(\mathcal{A}_*)$ onde

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



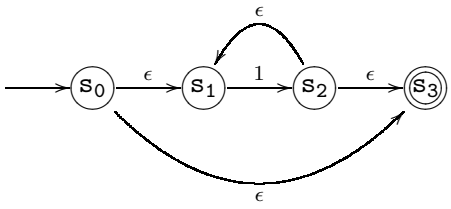
e a construção que verifica as condições impostas.

Os exemplos seguintes (4.8 e 4.9) sugerem algumas das vantagens de se estudar as demonstrações das proposições dadas no curso! Mais ainda, quando essas provas descrevem *algoritmos* (ou métodos) para resolver problemas... Convém no entanto notar que nem sempre é mais fácil nem menos trabalhoso seguir o método sugerido na prova.

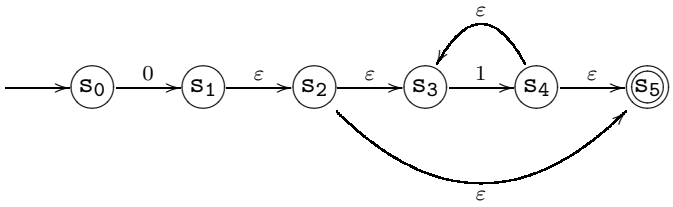
Exemplo 4.8. *Usando o método descrito, construir um AFND-ε para a expressão regular* $01^\star + 1$.

Os autómatos para 1 *e* 0*, são trivais:*

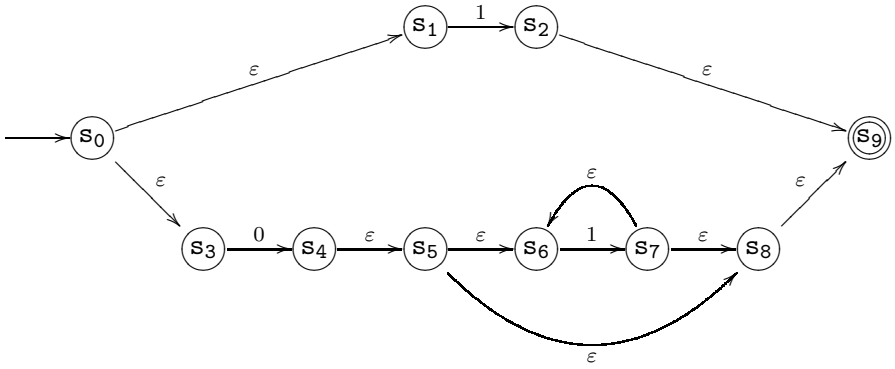
Para $1^\star$ *vem*



e a concatenação $01^\star$ *resulta em:*



e a reunião $01^\star + 1$ *vem:*



Exercício 4.4. *Por eliminação de transições-ε, obter um AFD equivalente.* $\diamond$

Resolução 4.4

Começamos por determintar o fecho por transições por ε, de cada estado do AFND:

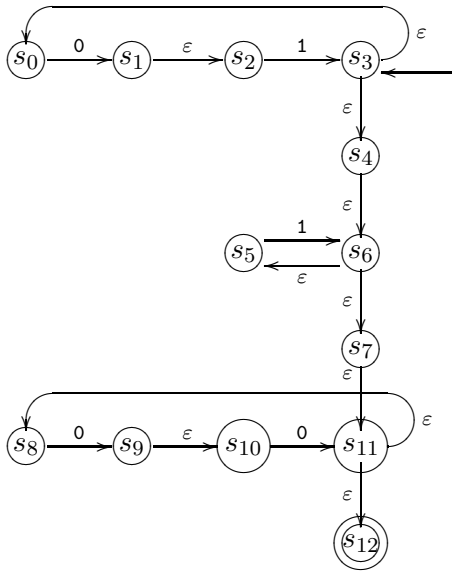
$$\begin{aligned} Fecho_\varepsilon(s_0) &= \{s_0, s_1, s_3\} \\ Fecho_\varepsilon(s_2) &= \{s_2, s_9\} \\ Fecho_\varepsilon(s_4) &= \{s_4, s_5, s_6, s_8, s_9\} \\ Fecho_\varepsilon(s_5) &= \{s_5, s_6, s_8, s_9\} \\ Fecho_\varepsilon(s_7) &= \{s_7, s_6, s_8, s_9\} \\ Fecho_\varepsilon(s_8) &= \{s_8, s_9\} \end{aligned}$$

e para os restantes temos $Fecho_\varepsilon(s) = \{s\}$.

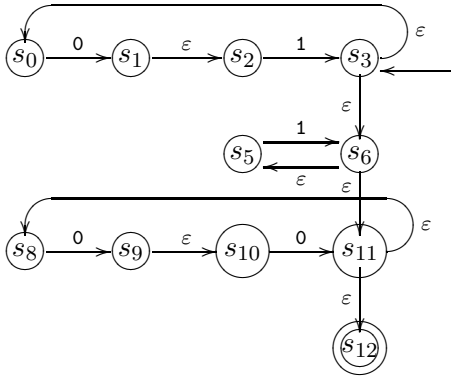
O autómato determinístico equivalente é definido pela tabela seguinte:

δ_D	1	0
$\rightarrow \{s_0, s_1, s_3\}$	$\{s_2, s_9\}$	$\{s_4, s_5, s_6, s_8, s_9\}$
$\star\{s_2, s_9\}$	$\emptyset$	$\emptyset$
$\star\{s_4, s_5, s_6, s_8, s_9\}$	$\{s_7, s_6, s_8, s_9\}$	$\emptyset$
$\star\{s_7, s_6, s_8, s_9\}$	$\{s_7, s_6, s_8, s_9\}$	$\emptyset$

Exemplo 4.9. *Seja* $L = L(\ (01)^\star 1^\star (00)^\star\ )$. *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 segundo a prova da Proposição 4.2 para* L *é*



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.

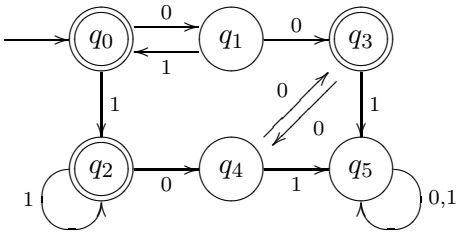


Seguindo a ideia da prova da Proposição 3.4, vamos determinar um autómato finito determinístico equivalente a este último.

O estado inicial desse AFD vai corresponder ao $Fecho_\varepsilon(s_3)$ *(ou seja, ao conjunto dos estados acessíveis do estado inicial do AFND-*ε*), sendo* $\{s_3, s_0, s_6, s_5, s_{11}, s_8, s_{12}\}$*. As transições são as que constam da tabela seguinte, a coluna mais à esquerda indicando o estado o autómato está.*

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

Denotando 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, *respectivamente, o AFD que obtivemos é o representado abaixo.*



Por sorte, *o autómato que obtivemos é o autómato determinístico mínimo (isto é, com o menor número de estados e que não encrava) que aceita* $\mathcal{L}(\ (01)^\star 1^\star (00)^\star\ )$.

Exercício 4.4.1. Tente justificar que o AFD obtido no exemplo anterior é o AFD mínimo, analisando o que cada estado desse autómato está a memorizar sobre palavra que é lida.

Exercício 4.5. *Constrói um autómato finito não-determinístico que reconheça cada uma das linguagens definidas pelas seguintes expressões regulares:*

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

b) $(aa)^\star bbb(bbb)^\star$

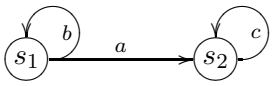
c) $(11 + 0)^\star(00 + 1)^\star$

◇

4.2.2 De Autómatos Finitos para Expressões Regulares

Proposição 4.3. *Qualquer que seja a linguagem L aceite por um autómato finito N, existe uma expressão regular r tal que $L(r) = L$.*

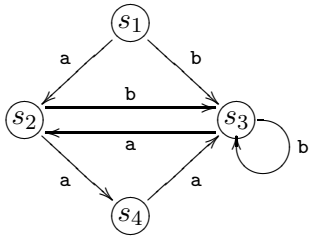
Exemplo 4.10. *Considere o seguinte diagrama:*



É fácil verificar que:

- $b^\star$ *descreve a linguagem determinada por caminhos de* s_1 *para* s_1 *(i.e.,* $\widehat{\delta}(s_1, b^\star) = s_1$*)*
- $c^\star$ *descreve a linguagem determinada por caminhos de* s_2 *para* s_2
- $\emptyset$ *descreve a linguagem determinada por caminhos de* s_2 *para* s_1 *(porquê?)*
- $b^\star ac^\star$ *descreve a linguagem determinada por caminhos de* s_1 *para* s_2

Exemplo 4.11. *Será ainda possível descrever por uma expressão regular cada uma das linguagens das palavras que levam o autómato do estado s_1 a cada um dos restantes estados?*



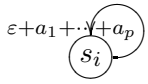
Prova. (Proposição 4.3, Método 1)

Seja L a linguagem aceite pelo autómato $\mathcal{A} = (\{s_1, \ldots, s_n\}, \Sigma, \delta, s_1, F)$ e considere-se o seu diagrama de transição.

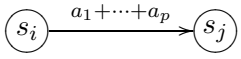
A prova que vamos fazer é válida quer $\mathcal{A}$ seja um AFD, um AFND, ou um AFND-ε.

Notar que estamos a supor que os estados são numerados de 1 a $|\mathcal{S}|$, sendo 1 o número que se atribui ao estado inicial. A expressão regular que descreve a linguagem determinada por caminhos generalizados do estado s_i para o estado s_j pode ser obtida por recorrência do modo seguinte. Seja r_{ij}^k a expressão regular que descreve a sublinguagem determinada pelos caminhos de s_i para s_j que passam quando muito por estados intermédios etiquetados com números não superiores a k. Iremos associar a cada recorrência um diagrama que tenta ilustrar os caminhos, mas que não representa um autómato finito. Temos, assim,

$$r_{ii}^0 = \begin{cases} \varepsilon & sse \text{ não existem lacetes em } s_i \\ \varepsilon + a_1 + \cdots + a_p & sse \text{ os lacetes em } s_i \text{ estão etiquetados com } a_1, \ldots, a_p \end{cases}$$



$$r_{ij}^0 = \begin{cases} \emptyset & sse \text{ não existe arco c om origem em } s_i \text{ e fim em } s_j \\ a_1 + \cdots + a_p & sse \text{ } a_1, \ldots, a_p \text{ etiquetam os arcos com origem em } s_i \text{ e fim em } s_j \end{cases}$$

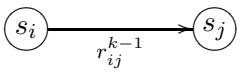


Define-se r_{ij}^k para $k \geq 1$ por recorrência:

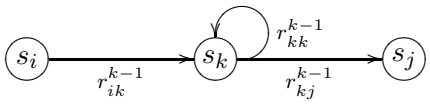
$$r_{ij}^k = r_{ij}^{k-1} + r_{ik}^{k-1}(r_{kk}^{k-1})^\star r_{kj}^{k-1}.$$

Esta definição traduz o facto dos caminhos generalizados de s_i para s_j passando quando muito por estados numerados até k serem:

- caminhos de s_i para s_j passando quando muito por estados numerados até $k-1$.



- obtidos por concatenação dum qualquer caminho de s_i para s_k que não tenha s_k como estado intermédio, com um ou nenhum ciclo s_k para s_k (sem passar por estados de número superior ou igual a s_k), e com um caminho qualquer de s_k para s_j que não passe por s_k.

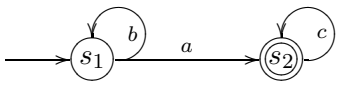


Pode concluir-se, por definição de expressões regulares, que r_{ij}^k é sempre uma expressão regular, qualquer que seja $k \in \mathbb{N}$. Para $n = |\mathcal{S}|$, a expressão regular r_{ij}^n descreve a linguagem determinada pelos caminhos generalizados de i para j (que podem passar por qualquer estado do autómato). A expressão regular da linguagem que o autómato reconhece é

$$\sum_{f \in \{\text{números dos estados finais}\}} r_{1f}^n$$

□

Exemplo 4.12. *Para o mesmo diagrama de transição do Exemplo 4.10, vamos agora determinar as expressões regulares das linguagens que levam o autómato do estado s_i ao estado s_j, com $i, j \in \{1, 2\}$ seguindo a prova da Proposição 4.3.*



$$r_{11}^0 = \varepsilon + b \qquad r_{22}^0 = \varepsilon + c \qquad r_{12}^0 = a \qquad r_{21}^0 = \emptyset$$
$$r_{11}^1 = r_{11}^0 + r_{11}^0 (r_{11}^0)^\star r_{11}^0 = \varepsilon + b + (\varepsilon + b)(\varepsilon + b)^\star(\varepsilon + b) = b^\star$$
$$r_{22}^1 = r_{22}^0 + r_{21}^0 (r_{11}^0)^\star r_{12}^0 = \varepsilon + c + \emptyset(\varepsilon + b)^\star a = \varepsilon + c$$
$$r_{12}^1 = r_{12}^0 + r_{11}^0 (r_{11}^0)^\star r_{12}^0 = a + (\varepsilon + b)(\varepsilon + b)^\star a = b^\star a$$
$$r_{21}^1 = r_{21}^0 + r_{21}^0 (r_{11}^0)^\star r_{11}^0 = \emptyset$$
$$r_{11}^2 = r_{11}^1 + r_{12}^1 (r_{22}^1)^\star r_{21}^1 = r_{11}^1 = b^\star$$
$$r_{12}^2 = r_{12}^1 + r_{12}^1 (r_{22}^1)^\star r_{22}^1 = b^\star a + b^\star a(\varepsilon + c)^\star(\varepsilon + c) = b^\star a c^\star$$
$$r_{22}^2 = r_{22}^1 + r_{22}^1 (r_{22}^1)^\star r_{22}^1 = c^\star \qquad r_{21}^2 = \emptyset$$

$$L(\mathcal{A}) = L(r_{12}^2) = L(b^\star a c^\star)$$

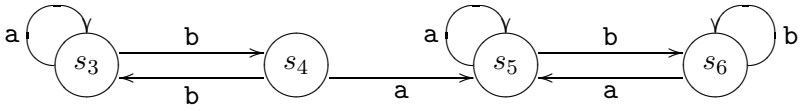
Não é difícil concluir que este pode ser um processo (muito) lento para determinar uma expressão regular que descreva a linguagem que um dado autómato reconhece, principalmente se o autómato for bastante simples. Temos, que, as desvantagens deste método são:

- determinar $O(n^3)$ expressões
- expressões regulares muito grandes $O(4^n)$

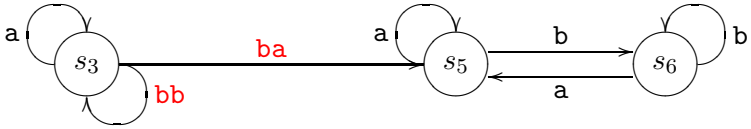
4.2.2.1 Obtenção de expressões regulares por eliminação de estados

Se o objectivo for a determinação duma expressão regular que defina a linguagem que um dado autómato finito reconhece, é mais simples usar **o método da "eliminação de estados".** A ideia é eliminar sucessivamente cada um dos estados do autómato, etiquetando, com expressões regulares, as transições sucessivamente introduzidas para substituição das que envolvem cada estado eliminado.
Para melhor compreensão do método, começamos por analisar o exemplo seguinte. Notar que os diagramas apresentados não representam autómatos finitos.

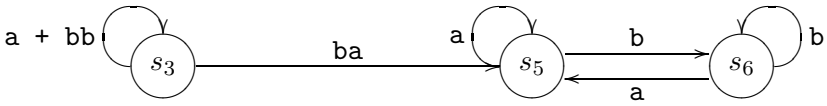
Exemplo 4.13. *Determinar uma expressão regular que descreva a linguagem das palavras que levam o autómato do estado s_3 ao estado s_6, considerando o seguinte diagrama de transição.*



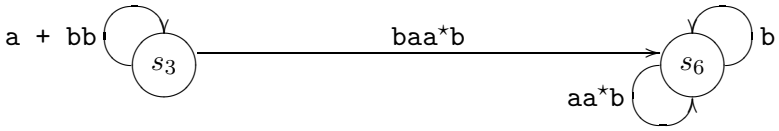
Para eliminar s_4, deve-se substituir o caminho $s_3s_4s_3$ por um lacete em s_3 etiquetado por `bb`*, e o caminho $s_3s_4s_5$ por um arco de s_3 para s_5 etiquetado por* `ba`*. Obtem-se o diagrama seguinte,*



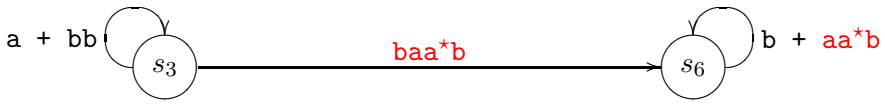
o qual podemos ainda representar do modo seguinte:



Se procedermos à eliminação de s_5 obtemos



ou equivalentemente



concluindo-se que a expressão regular

$$(\mathtt{a} + \mathtt{bb})^{\star}\mathtt{baa}^{\star}\mathtt{b}(\mathtt{b} + \mathtt{aa}^{\star}\mathtt{b})^{\star}$$

descreve a linguagem das palavras que levam o autómato dado do estado s_3 ao estado s_6.

Método de eliminação de estados:

1. Começar por verificar se o autómato dado tem estados que não sejam acessíveis do estado inicial. Se tiver, retirá-los bem como as transições que tenham origem ou fim nesses estados.

2. Caso existam transições para o estado inicial, pode-se criar um novo estado inicial e uma transição por ε do novo estado inicial para o antigo. Este passo não é **obrigatório** mas simplifica a obtencao da expressão regular.

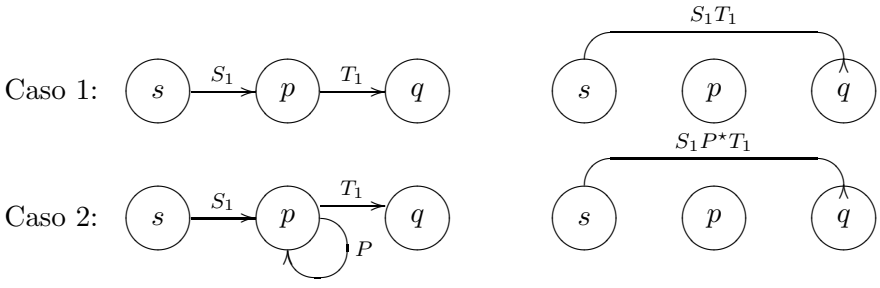
3. Proceder de modo análogo a (1) para eliminar todos os estados que não permitam chegar a algum dos estados finais.

4. Se depois de ter efectuado (1) e (3) já não existirem estados finais, então dar como resultado a expressão $\emptyset$.

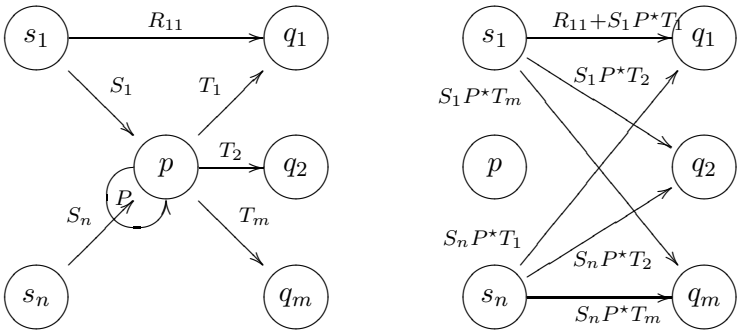
5. Se depois de ter efectuado (1) e (3) ainda existir algum estado final, então introduzir um novo estado que passa a ser o único estado final, e introduzir transições por ε de cada estado, que era final no autómato dado, para o novo estado final. Marcar os estados que no autómato dado eram finais como não finais.

6. Eliminar sucessivamente cada um dos estados do autómato obtido em (5) com excepção do estado inicial e do estado final.

   Não pretendendo ser formais nesta exposição, vamos dar uma ideia através de esquemas das transformações a efectuar para eliminar um estado p. Quando analisar as figuras, considere que s e q podem ser o mesmo estado (pelo que nessa situação, deviam ter sido representados por um único vértice).



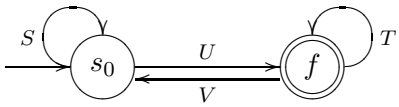
Caso Geral:



Sejam s_i, para $i = 1, \ldots n$ os predecessores do estado p e q_j, para $j = 1, \ldots m$ os sucessores de p. Seja S_i a expressão regular que etiqueta o arco do estado s_i para p, T_j a expressão regular que etiqueta o arco de p para q_j, R_{ij} a expressão regular que etiqueta o arco de s_i para q_j e P a expressão regular que etiqueta o arco de p para p.
Para eliminar o estado p, substituir R_{ij} por $R_{ij} + S_iP^\star T_j$, para $i = 1, \ldots, n$, $j = 1, \ldots, m$ e eliminar todas as transições de e para p.
Notar que muitas das expressões regulares poderão ser $\emptyset$.

- Se $P = \emptyset$, $P^\star = \varepsilon$ e $S_iP^\star T_j = S_iT_j$.
- Se S_i ou T_j for $\emptyset$, $S_iP^\star T_j = \emptyset$.
- Se $R_{ij} = \emptyset$ e $S_iP^\star T_j \neq \emptyset$, acrescentar um arco de s_i para q_j e etiquetá-lo com $S_iP^\star T_j$

Depois de aplicar o método, resulta um autómato da forma

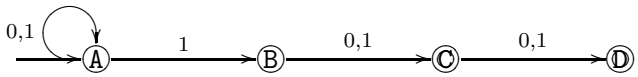


podendo algumas das expressões regulares serem $\emptyset$.
A expressão regular é obtida considerando as expressões regulares:

- para as palavras que levam do estado inicial ao final, a primeira vez: $S^\star U$
- para as palavras que levam do estado final ao estado final: $(T + VS^\star U)^\star$

Donde resulta, por exemplo, a expressão, $S^\star U(T + VS^\star U)^\star$
Se o estado inicial é o estado final a expressão regular é $S^\star$. Se não houverem transições para o estado inicial e transições dos estados finais, então a expressão regular é apenas U. Isto acontece se se aplicarem sempre os passos (2) e (5).

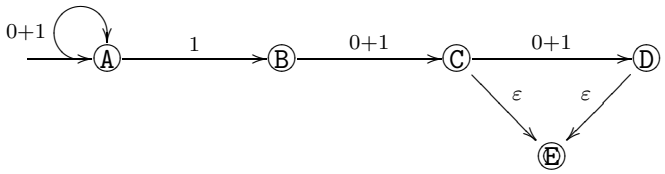
Exemplo 4.14. *Considere o autómato $\mathcal{A}$ cujo diagrama de transição é o seguinte:*



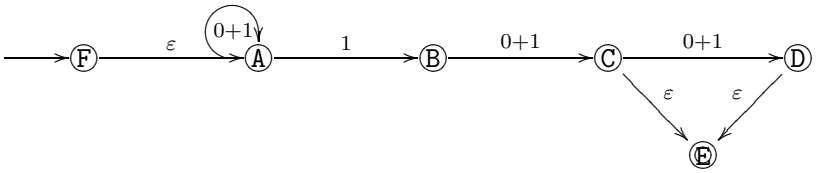
A linguagem aceite por $\mathcal{A}$ *é*

$$L = \{x \in \{0,1\}^\star \mid x \text{ tem um } 1 \text{ na penúltima ou na antepenúltima posição}\}$$

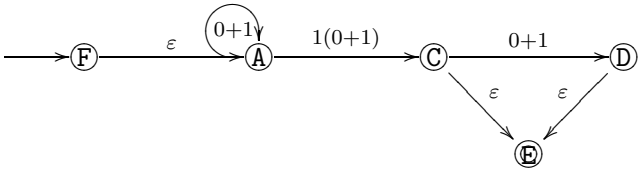
Para aplicar o método da eliminação de estados, vamos acrescentar o estado final e substituir as etiquetas por expressões regulares.



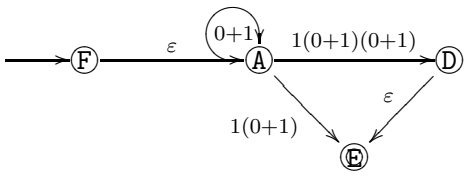
Podemos também acrescentar um novo estado inicial:



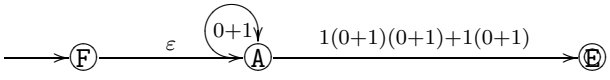
Eliminando `B` *vem, notando que* $1\emptyset^\star(0+1) \equiv 1(0+1)$*,*



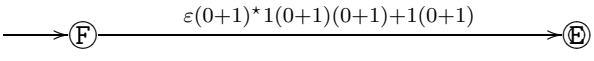
Eliminando `C` *vem,*



Eliminando `D` *vem,*



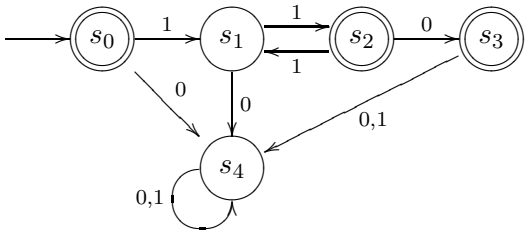
Eliminando `A` *vem,*



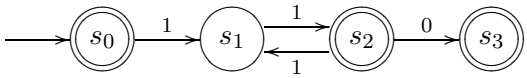
e finalmente, notando que $\varepsilon r \equiv r$ para qualquer expressão regular r, uma expressão regular que descreve L é

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

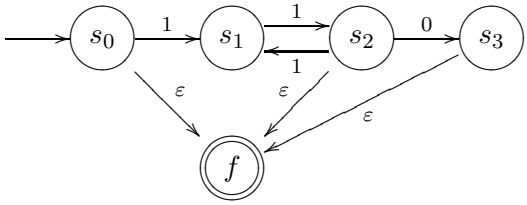
Exemplo 4.15. *Pretende-se determinar, pelo método de eliminação de estados, uma expressão regular que descreva a linguagem reconhecida pelo autómato seguinte.*



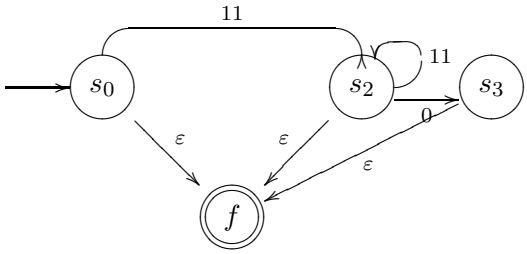
De acordo com (3), eliminamos s_4:



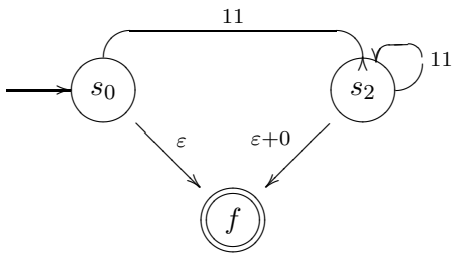
Transformamos o autómato, segundo (5):



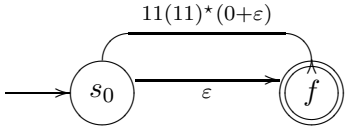
Eliminamos agora eliminamos s_1:



Em seguida, eliminamos s_3:



E finalmente, eliminamos s_2:



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

Sumário Podemos sumariar a caracterização das linguagens regulares:

Dada uma linguagem $L \subseteq \Sigma^\star$, é equivalente dizer

- "L é regular", ou dizer
- "L é aceite por um autómato finito determinístico", ou dizer
- "L é aceite por um autómato finito não determinístico", ou ainda
- "L é aceite por um autómato finito não determinístico com transições-ε".

4.3 Propriedades das linguagens regulares

Nesta secção vamos estudar algumas propriedades das linguagens regulares.

Capítulo 5

Propriedades das Linguagens Regulares

Considerando um alfabeto, já vimos que podemos caracterizar a classe das linguagens regulares sobre esse alfabeto como o conjunto das linguagens que podem ser descritas por expressões regulares ou, equivalentemente, aceites (reconhecidas) por algum autómato finito.
Vamos estudar, neste capítulo, outros resultados fundamentais sobre esta classe de linguagens, nomeadamente:

- A classe das linguagens regulares é fechada para operações sobre linguagens como a união finita, a complementação, a intersecção, a concatenação (finita) e o fecho de Kleene.
- Existem linguagens que não são regulares e podemos demonstrar que uma linguagem não é regular mostrando que não obedece a uma condição de *repetição* que as linguagens regulares estão sujeitas.
- É possível decidir se dois autómatos são equivalentes, isto é, se reconhecem a mesma linguagem. Em particular, dado qualquer autómato finito, podemos obter um autómato finito determinístico equivalente com o menor número de estados. Esta propriedade, é uma das mais importantes das linguagens regulares.
- Existem algoritmos (alguns eficientes) para responder a alguns problemas de decisão sobre linguagens regulares. Problemas como: determinar se uma palavra pertence a uma dada linguagem regular; determinar se uma linguagem regular é vazia ou $\Sigma^\star$; determinar se duas linguagens regulares se intersectam, se são equivalentes, se uma está contida na outra, etc.

5.1 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 5.1. *O conjunto dos números reais não negativos, é fechado para a soma mas não é fechado para a subtracção, pois*

$$\forall x \in \mathbb{R}_0^+ \, \forall y \in \mathbb{R}_0^+ \;\; x + y \in \mathbb{R}_0$$

e como, por exemplo, $3 - 5 \notin \mathbb{R}_0$*, sabemos que*

$$\exists x \in \mathbb{R}_0^+ \, \exists y \in \mathbb{R}_0^+ \;\; x - y \notin \mathbb{R}_0$$

ou seja que não é verdade que

$$\forall x \in \mathbb{R}_0^+ \, \forall y \in \mathbb{R}_0^+ \;\; x - y \in \mathbb{R}_0$$

Dizer que

> A classe de linguagens regulares é fechada para a união finita, a complementação, a intersecção, a concatenação (finita) e o fecho de Kleene (entre outras operações).

é dizer que

> A união finita de linguagens regulares é regular, a linguagem complementar duma linguagem regular é regular, o fecho de Kleene duma linguagem regular é regular, ...

Proposição 5.1. *A classe das linguagens regulares é fechada para*

- ***concatenação*** *finita*
- ***fecho de Kleene***
- *a* ***reunião*** *finita*
- *a* ***complementação***
- *a* ***intersecção*** *finita*
- *a* ***inversão***

Prova. Dado que uma linguagem regular é descrita por uma expressão regular então é fácil concluir que as linguagens regulares são fechadas para a concatenação (finita), o fecho de Kleene e a união finita.

Concatenação Se L e M são duas linguagens regulares então existem expressões regulares r e s tal que $L = L(r)$ e $M = L(s)$. Então pela definição do operador $\cdot$, $L(rs) = LM$, isto é, rs é a expressão regular que descreve a concatenação de L e M.

Fecho de Kleene Se L é uma linguagem regular então existe uma expressão regular r tal que $L = L(r)$. Então pela pela definição do operador $\star$, $L(r^\star) = L(r)^\star$, isto é, $r^\star$ é a expressão regular que descreve o fecho de Kleene de L.

Reunião 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* $2 \leq i \leq n$.

Base. Se L_1 e L_2 são regulares, sejam r_1 e r_2 expressões regulares que as descrevem. Então, $r_1 + r_2$ descreve $L_1 \cup L_2$, isto é $L(r_1 + r_2) = L_1 \cup L_2$, pelo que $L_1 \cup L_2$ é regular.

Indução. Dado $n \geq 2$, suponhamos, como hipótese de indução, que quaisquer que sejam $L_1, \ldots, L_n$ linguagens regulares de alfabeto Σ, a reunião $L_1 \cup \ldots \cup L_n$ 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 concluímos que $Q_1 \cup \ldots \cup Q_n \cup Q_{n+1}$ é reunião de duas linguagens regulares. Como mostrámos que a reunião de *duas* linguagens regulares é regular, concluímos que $Q_1 \cup \ldots \cup Q_n \cup Q_{n+1}$ é regular.

Complementação ou seja, se L é uma linguagem regular sobre Σ, então $\overline{L} = \Sigma^\star \setminus L$ também é uma linguagem regular. Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um AFD em que δ é uma função total de $S \times \Sigma$ em S e tal que $L = \mathcal{L}(\mathcal{A})$. O autómato $\mathcal{A}' = (S, \Sigma, \delta, s_0, S \setminus F)$ é um AFD que aceita $\Sigma^\star \setminus L$, pelo que $\Sigma^\star \setminus L$ é regular.

Intersecção Sejam L e M são duas linguagens regulares. Como $L \cap M = \overline{\overline{L} \cup \overline{M}}$, concluímos usando 5.1 e 5.1 que $L \cap M$ é uma linguagem regular.

Inversão , ou seja se L é uma linguagem regular L^R, constituída pelas palavras inversas das de L, é regular. Podemos provar por indução na estrutura da expressão regular r que descreve $L + L(r)$.

Base. Se r é $\emptyset$, ϵ ou a, $a \in \Sigma$ então r^R é r

Indução.

$r = r_1 + r_2$, então $r^R = r_1^R + r_2^R$.
$r = r_1 r_2$, então $r^R = r_2^R r_1^R$.
$r = r_1^\star$, então $r^R = (r_1^R)^\star$.

□

Vimos anteriormente que dados dois autómatos finitos $\mathcal{A}_1$ e $\mathcal{A}_2$, se pode definir facilmente um autómato finito (mais precisamente, um AFND-ε) que reconhece $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. Se se pretender determinar um AFD que reconheça $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$, podemos ter que encontrar o AFD equivalente àquele que obtivemos pelo processo referido, o que pode ser algo trabalhoso! Pode o leitor, não se ter apercebido ainda que 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 $L_2 = \mathcal{L}(\mathcal{A}_2)$.

Começar por encontrar AFDs que não encravem e que sejam equivalentes a $\mathcal{A}_1$ e a $\mathcal{A}_2$. Para obter $\overline{L_1}$, basta considerar no AFD respectivo, que os estados finais passam a ser não-finais e os não-finais passam a finais. Deve proceder do mesmo modo para obter o AFD que reconhece $\overline{L_2}$.

Depois disto, determina-se um autómato finito que reconhece a $\overline{L_1} \cup \overline{L_2}$, o qual terá que ser um AFD que não encrave, uma vez que por fim, queremos determinar o autómato que reconhece $\overline{\overline{L_1} \cup \overline{L_2}}$ (ou seja, a linguagem complementar de $\overline{L_1} \cup \overline{L_2}$).

5.1.1 Autómatos produto

Mas também se pode obter um autómato finito para a intersecção (reunião e outras operações) de duas linguagens regulares, pela construção do chamado *autómato produto.*
Supomos disponíveis AFDs

$$\mathcal{A}_1 = (S_2, \Sigma, \delta_1, i_1, F_1)$$

e

$$\mathcal{A}_2 = (S_2, \Sigma, \delta_2, i_2, F_2)$$

tais que δ_1 e δ_2 são totais (isto é, os autómatos não encravam), os quais reconhecem respectivamente L_1 e L_2. Podemos então dizer que, qualquer que seja $x \in \Sigma^\star$:

- $x \in L_1 \cap L_2$ 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;

Consideramos um autómato que vai, por assim dizer, *simular a execução simultanea de* $\mathcal{A}_1$ *e* $\mathcal{A}_2$. Os seus estados são pares (s_1, s_2) com $s_1 \in S_1$ e $s_2 \in S_2$, e a função de transição δ é dada por

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

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).
Designamos genericamente este autómato por $\mathcal{A}_{\surd} = (S_1 \times S_2, \Sigma, \delta_p, (i_1, i_2), F_p)$
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 $L_1 \cup L_2$;

Exercício 5.1. *Mostre por indução em* $|x|$, *que* $\forall\, x \in \Sigma^\star, \ \widehat{\delta_p}((s_1, s_2), x) = (\widehat{\delta_1}(s_1, x), \widehat{\delta_2}(s_2, x))$
$\diamond$

Proposição 5.2. *Seja* $\mathcal{A}_\cap$ *o autómato produto para a intersecção* $L(\mathcal{A}_\cap) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$

Dem:

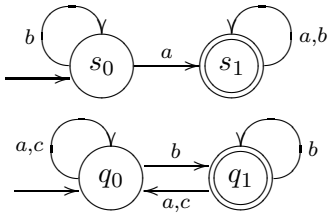
$$\begin{aligned}
x \in L(\mathcal{A}_\cap) &\iff \widehat{\delta_\cap}(i_\cap, x) \in F_\cap \\
&\iff \widehat{\delta_\cap}((i_1, i_2), x) \in F_1 \times F_2 \\
&\iff (\widehat{\delta_1}(i_1, x), \widehat{\delta_2}(i_2, x)) \in F_1 \times F_2 \\
&\iff \widehat{\delta_1}(i_1, x) \in F_1 \ \wedge \ \widehat{\delta_2}(i_2, x) \in F_2 \\
&\iff x \in L(\mathcal{A}_1) \ \wedge \ x \in L(\mathcal{A}_2) \\
&\iff x \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)
\end{aligned}$$

□

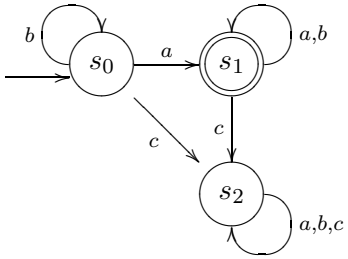
Exercício 5.2. *Mostre que* $L(\mathcal{A}_\cup) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ $\diamond$

Exercício 5.3. *Como é que analisando o autómato produto se pode saber se* $\mathcal{L}(\mathcal{A}_1) \subset \mathcal{L}(\mathcal{A}_2)$*? E saber se* $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$*? E ainda, se* $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$*?* $\diamond$

Exemplo 5.2. *Para ilustrar o método que acabámos de expor, supomos dados os autómatos finitos seguintes.*



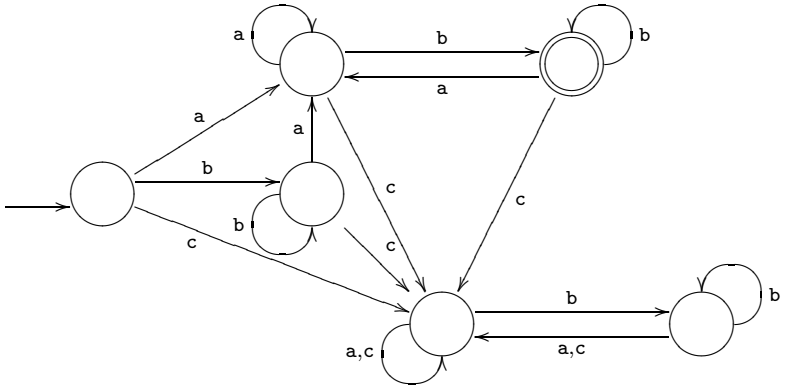
A intersecção das linguagens que estes autómatos reconhecem é o conjunto das palavras de alfabeto $\{\mathtt{a},\mathtt{b},\mathtt{c}\}$ *que têm algum* `a`*, terminam em* `b` *e não têm* `c`*'s. Note que os autómatos não têm alfabetos iguais, pelo que antes de aplicar o processo, vamos transformar o primeiro autómato.*



As transições do autómato que reconhece a intersecção são dadas na tabela seguinte, sendo o seu conjunto de estados finais $\{(s_1,q_1)\}$.

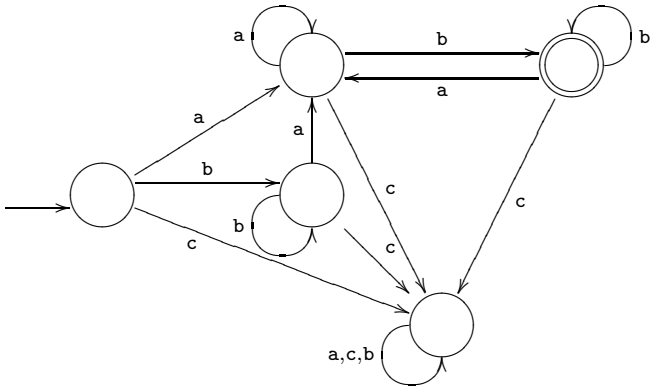
	a	b	c
(s_0,q_0)	(s_1,q_0)	(s_0,q_1)	(s_2,q_0)
(s_1,q_0)	(s_1,q_0)	(s_1,q_1)	(s_2,q_0)
(s_0,q_1)	(s_1,q_0)	(s_0,q_1)	(s_2,q_0)
(s_2,q_0)	(s_2,q_0)	(s_2,q_1)	(s_2,q_0)
(s_1,q_1)	(s_1,q_0)	(s_1,q_1)	(s_2,q_0)
(s_2,q_1)	(s_2,q_0)	(s_2,q_1)	(s_2,q_0)

Tal autómato pode ser representado pelo diagrama seguinte

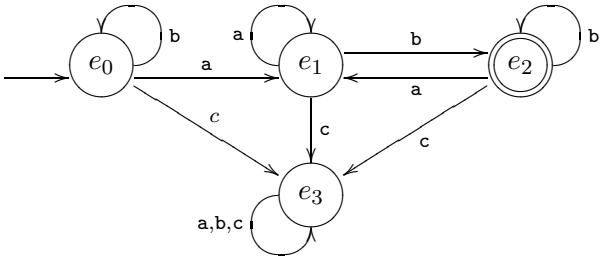


concluindo-se facilmente que não é mínimo. Por exemplo, o AFD seguinte é equivalente ao

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, é bem menos trabalhoso "pensar um pouco" do que usar o processo descrito anteriormente. O AFD que não encrava e que tem o menor número de estados é o seguinte.*



O que cada estado memoriza...

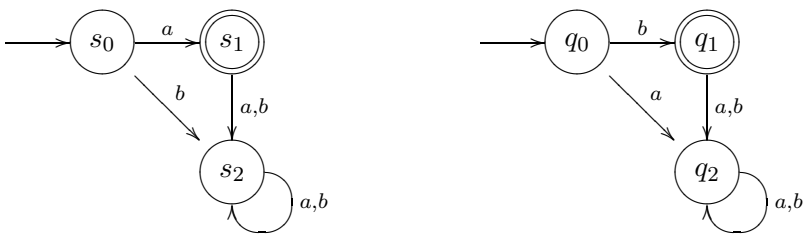
e_0**:** *"a palavra ainda não tem* `a` *nem tem* `c`*";*

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

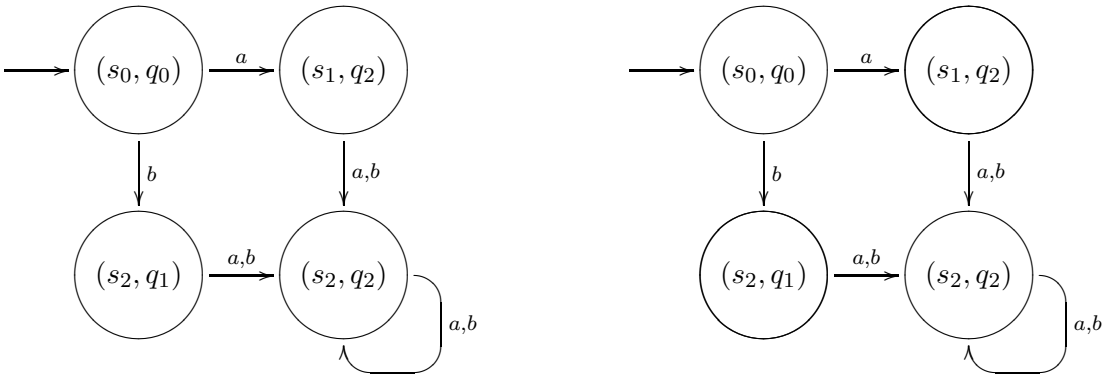
e_2**:** *"a palavra tem algum* `a`*, termina em* `b` *e não tem* `c`*'s";*

e_3**:** *"a palavra tem algum* `c`*".*

Exemplo 5.3. *Considere os dois autómatos seguintes.*



Usando o processo descrito anteriormente obtêm-se os dois autómatos seguintes, o da esquerda reconhecendo a $\{a\} \cap \{b\}$ (ou seja, $\emptyset$) e o da direita $\{a\} \cup \{b\}$.



Claramente, este não é o melhor processo para determinar autómatos que definam $\emptyset$ e $\{a, b\}$!

5.1.2 Linguagens Finitas

Exemplo 5.4. *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, pois 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 $(\mathtt{a}+\mathtt{b}(\mathtt{a}+\mathtt{b})^\star)(\mathtt{0}(\mathtt{a}+\mathtt{b})(\mathtt{a}+\mathtt{b})(\varepsilon+\mathtt{a}+\mathtt{b}))^\star\mathtt{0}$, respectivamente.
A linguagem $\overline{L_3} \cap (\ \{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\}\{\mathtt{0}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^\star\}\ )$ é regular.
Como L_3 é regular, também $\overline{L_3}$.

Por outro lado, $\{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\}$ 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 $k \geq 0$. Note 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.

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

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

Exercício 5.3.1. Justifique 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. Conclua que $\{\mathtt{b}^n\mathtt{a}^n \mid n \leq 1000\}$ é regular pois é finita.

Exercício 5.3.2. Sejam L_1 e L_2 linguagens de alfabeto Σ tais que $L_1 \cap L_2 = \emptyset$, e L_2 é regular.

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

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

A propósito do exercício anterior, note que a condição $L_1 \cap L_2 = \emptyset$ é fundamental. De facto, se $L_1 \cap L_2 \neq \emptyset$, 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 considerar que L_2 é $\Sigma^\star$ e que $L_1 = \{a^n \mid n \text{ primo}\}$, sendo a algum elemento de Σ. Na secção seguinte mostra-se que L_1 não é regular.

5.2 Existência de Linguagens que não são Regulares

Exemplo 5.5. *Seja $\Sigma = \{\mathtt{0},\mathtt{1}\}$ e seja $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$. Vamos mostrar que L não é regular, justificando a impossibilidade de construir um autómato finito que a aceite.*

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

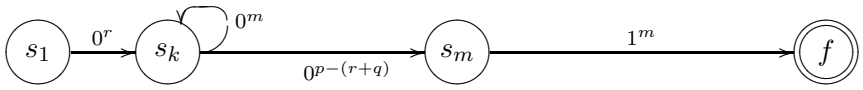
- *s_m é o estado a que o autómato chega depois de consumir $\mathtt{0}^p$, partindo de s_1;*
- *f é o estado a que chega depois de consumir $\mathtt{1}^p$, partindo de s_m.*

$$\overbrace{000000000000000000000}^{p}\overbrace{111111111111111111111}^{p}$$
$$\uparrow$$
$$s_m$$

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

- *partindo do estado inicial, o autómato consome* $\mathtt{0}^r$*, para um certo* r *tal que* $0 \leq r < p$ *e chega pela primeira vez a* s_k *(note que, se* s_k *for o estado inicial, então aqui estamos a considerar* $r = 0$*, ou seja que não consumiu ainda nada);*
- *partindo de* s_k *consome* $\mathtt{0}^q$*, 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* $\mathtt{0}$*'s ou parte de* s_k *consome* $\mathtt{0}^{p-(r+q)}$ *atingindo* s_m*.*

Esquematizando[1]*...*



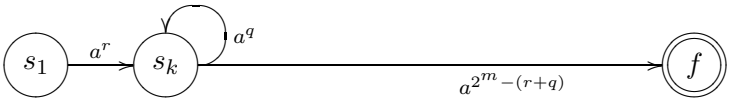
Mas se isto acontecesse, também a palavra $\mathtt{0}^r\mathtt{0}^{p-(r+q)}\mathtt{1}^p$*, ou seja,*

$$\mathtt{0}^{p-q}\mathtt{1}^p$$

seria aceite pelo autómato. O que é absurdo, pois sendo $q \geq 1$ *tem-se* $\mathtt{0}^{p-q}\mathtt{1}^p \notin L$*. O absurdo resultou de se ter suposto que existia um autómato finito que reconhecia* L*. Portanto,* L *não é aceite por qualquer autómato finito.*
Note ainda que não só $\mathtt{0}^{p-q}\mathtt{1}^p$ *seria aceite como também* $\mathtt{0}^p\mathtt{1}^p$ *(claro!),* $\mathtt{0}^{p+q}\mathtt{1}^p$*,* $\mathtt{0}^{p+2q}\mathtt{1}^p$*,* $\mathtt{0}^{p+3q}\mathtt{1}^p$*,* $\mathtt{0}^{p+4q}\mathtt{1}^p$ *..., ou seja,*

$$\forall i \in \mathbb{N} \quad \mathtt{0}^r\mathtt{0}^{i\times q}\mathtt{0}^{p-(q+r)}\mathtt{1}^p \text{ seria aceite}$$

Exemplo 5.6. *Seja* $\Sigma = \{\mathtt{a}\}$ *e seja* $L = \{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\} = \{\mathtt{a}, \mathtt{a}^2, \mathtt{a}^4, \mathtt{a}^8, \mathtt{a}^{16}, \ldots\}$*. Vamos mostarr que* L *não é regular.*
Suponhamos, por contradição, que $L = L(\mathcal{A})$*. Seja* p *o número de estados de* $\mathcal{A}$*. Seja* $m \gg p$ *e consideremos a processamento de* $\mathtt{a}^{2^m}$*. Mais uma vez, ao processar os primeiros* m *símbolos terá de haver um estado que se repete: seja* s_k



onde $0 < q \leq m$ *mas então a palavra* a^{2^m+q} *também é aceite por* $\mathcal{A}$ *e* $2^m + q$ *não é potência de 2, pois temos:*

$$\begin{aligned} 2^m + q &\leq 2^m + m \\ &< 2^m + 2^m \\ &= 2^{m+1} \end{aligned}$$

[1]Convém notar, que s_1, s_k e s_m não têm que ser distintos dois a dois, como erradamente este esquema sugere!

e 2^{m+1} é a potência de 2 a seguir a 2^m! Mais uma vez o absurdo resultou de supor que L era regular.

Para o próximo exemplo, iremos necessitar de algumas noções de Teoria dos Números. Recordemos que, um inteiro não negativo é **primo** se e só se tiver dois e apenas dois divisores (ele próprio e a unidade).

Lema 5.1. *O conjunto dos inteiros não negativos que são primos é infinito.*

Prova. Suponhamos que só existiam n primos, sendo n um certo inteiro **fixo**. Ou seja, suponhamos que o conjunto dos primos era finito e que tinha exactamente n elementos, os quais podemos denotar por $p_1, p_2, \ldots, p_n$. Então, o inteiro positivo

$$1 + \prod_{i=1}^{n} p_i$$

não seria primo, já que é maior do que qualquer um dos primos $p_1, p_2, \ldots, p_n$.

Mas se $1+p_1p_2\ldots p_n$ não é primo, então algum dos primos o divide[2] (ou seja, é múltiplo de algum dos primos). Suponhamos que p_k divide $1 + p_1p_2\cdots p_n$.

Como p_k divide $p_1p_2\cdots p_n$, pode-se concluir que se p_k dividir $1 + p_1p_2\cdots p_n$ então p_k divide 1. (De facto, é bem conhecido que quaisquer que sejam x, y e z inteiros, se x dividir y e dividir $y + z$ então também divide z.)

Mas nenhum primo divide 1. O absurdo resultou de se ter suposto que o conjunto dos primos era finito. Logo, o conjunto dos primos é infinito. □

Exemplo 5.7. *A linguagem $L = \{0^p \mid p \text{ primo}\}$ não é regular.*

Vamos mostrar que não existe um autómato finito determinístico que aceite L. Na justificação precisamos de usar o facto do **conjunto de primos ser infinito** *que demonstrámos acima (c.f. Lema ??).*
Vamos supor que existia um tal autómato e vamos mostrar que se chega a uma contradição. Seja $\mathcal{A}$ um autómato finito determinístico que reconhece L, e seja n o número de estados de $\mathcal{A}$. Supomos que s_0 é o estado inicial.

Como o conjunto de primos é infinito, podemos dizer que existe um primo M tal que $M > 2n$ (pretendemos que M seja suficientemente grande quando comparado com n; basta que seja maior do que $2n$ como ficará claro mais à frente). Sendo $L = \mathcal{L}(\mathcal{A})$, o autómato reconheceria a palavra 0^M. Ora, $0^M = 0^n0^{M-n}$. Processar o prefixo 0^n envolve a passagem por $n + 1$ estados, e como $\mathcal{A}$ só tem n estados, terá que se passar duas vezes por algum estado – seja s_q por exemplo um estado que se repete. Como no exemplo anterior, podemos supor que

[2] Quaisquer que sejam os inteiros a e b, diz-se que a **divide** b se e só se existir um inteiro c tal que $b = a \times c$.

- *partindo do estado inicial o autómato consome* 0^i *para algum* i *tal que* $0 \leq i < n$ *e chega pela primeira vez a* s_q*;*
- *em seguida, partindo do estado* s_q *consome* 0^j *para algum* j *tal que* $0 \leq j < n$ *e* $i+j \leq n$*, e regressa pela primeira vez a* s_q*;*
- *finalmente, partindo de* 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. Escolhendo* k *convenientemente, podemos mostrar que existe uma dessas palavras que é aceite pelo autómato e que não pertence à linguagem* L*. Note que uma sequência de* 0*'s pertence a* L *se e só se tem comprimento primo. Mas, notando que* $(0^j)^k$ *é a palavra formada por repetição da palavra* 0^j *exactamente* k *vezes, temos*

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

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 factorização. . .), 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 reconhecia L*. Portanto,* L *não é aceite por qualquer autómato finito.*

5.2.1 Um lema da repetição para linguagens regulares

Para mostrar que uma dada linguagem L não é regular podemos proceder como nos exemplos **??** e **??**. Se supusermos que L é regular, podemos (nem sempre é possível) conseguir chegar a uma contradição, já que se L for regular terá que satisfazer a condição dada no Lema da Repetição, cuja ideia é a seguinte.

> Se L é uma linguagem regular qualquer e se L tiver palavras "suficientemente grandes", então *cada uma dessas palavras tem alguma subpalavra que pode ser*

repetida quantas vezes quisermos ou retirada, sendo a palavra resultante ainda uma palavra de L. O enunciado do lema esclarece o significado de "suficientemente grande". Note que nenhuma condição é imposta às palavras que não são "suficientemente grandes".

Proposição 5.3. (Lema da Repetição) *(Pumping Lemma)*
Seja L 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 $|x| \geq n$, então podemos escrever $x = uvw$ com $|uv| \leq n$ e $|v| \geq 1$, e tal que $uv^iw \in L$, para todo $i \geq 0$. Ou seja,

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

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

Prova. Seja L uma linguagem regular. Então existe um autómato finito que aceita L. Seja $\mathcal{A}$ um tal autómato, e seja n o seu número de estados. Sem perda de generalidade podemos supor que n não excede o número de estados de qualquer outro autómato que aceite L. Podemos ainda supor que o autómato não tem transições-ε, pois existiria um AFND equivalente com exactamente o mesmo número de estados (**Justificar!**). Seja x uma qualquer palavra de L, tal que $|x| \geq n$. Qualquer caminho no diagrama de transição de $\mathcal{A}$ correspondente ao processamento de x envolve $|x| + 1$ estados. Como $\mathcal{A}$ tem apenas n estados, um tal caminho terá que conter algum ciclo. Seja s_q o primeiro estado que se repete. 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^iw com $i \geq 0$. □

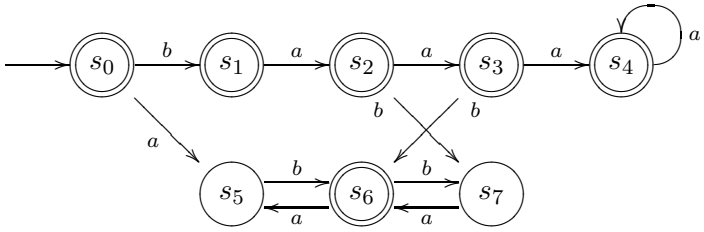
Exemplo 5.8. *Seja L a linguagem de alfabeto $\{\mathtt{a}, \mathtt{b}\}$ que é definida pela expressão $\mathtt{aa}(\mathtt{a}+\mathtt{b})^\star$. Vamos ver que podemos mostrar que, toda a palavra x em L que tenha pelo menos três símbolos, tem uma subpalavra que pode ser repetida quantas vezes se quiser ou retirada, obtendo-se uma palavra ainda da linguagem L.*
De facto, 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 for $x = \mathtt{aaa}x'$, então podemos considerar que a subpalavra que vamos repetir ou retirar é o

terceiro a. *Se for* $x = \texttt{aab}x'$, *então podemos considerar que a subpalavra que vamos repetir ou retirar é a formada pelo terceiro símbolo (ou seja* b*).*
Aquilo que estamos a ver é que se $x = \texttt{aaa}x'$ *e escolhermos* $u = \texttt{aa}$, $v = \texttt{a}$ *e* $w = x'$, *então* $uv^iw \in L$, *para todo* $i \geq 0$. *E, se for* $x = \texttt{aab}x'$ *e escolhermos* $u = \texttt{aa}$, $v = \texttt{b}$ *e* $w = x'$, *então* $uv^iw \in L$, *para todo* $i \geq 0$. *Note ainda que satisfizemos também as condições* $|v| \geq 1$, $|uv| \leq 3$ *e* $uvw = x$. *Ou seja, mostrámos que*

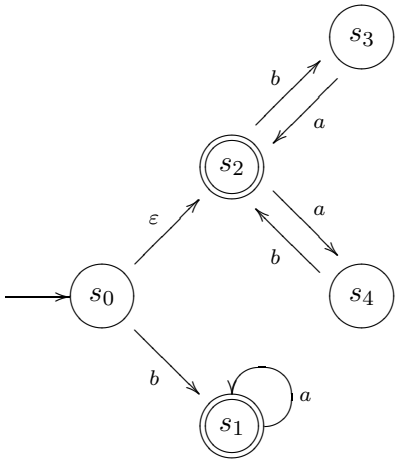
$$\exists n \in \mathbb{N}\ \forall x \in L\ \ (|x| \geq n \Longrightarrow \exists u \exists v \exists w\ \ (x = uvw \wedge v \neq \varepsilon \wedge |uv| \leq 1 \wedge (\forall i \in \mathbb{N}\ \ uv^iw \in L)))$$

tendo mostrado que podiamos tomar $n = 3$ *(verifique que há um autómato finito com três estados que aceita* L*).*

Exemplo 5.9. *A linguagem* L *de alfabeto* $\{\texttt{a}, \texttt{b}\}$ *que é definida pela expressão* $\texttt{ba}^\star + (\texttt{ab} + \texttt{ba})^\star$, *é reconhecida pelo autómato finito seguinte.*



Como esse autómato tem oito estados, podiamos escolher $n = 8$. *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*[3] L. *Tem-se,*

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

[3]Não esqueça que "reconhecer" significa que aceita todas, e apenas, as palavras de L.

Vamos agora mostrar que para cada x existem u, v e w nas condições do Lema. Se $x = \texttt{baaa}y$ então 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 $i \geq 0$. Note que, neste caso $y \in \{\texttt{a}\}^\star$.
Nos restantes casos, podemos escrever $x = x_1x_2x_3x_4y$ com $x_1x_2 \in \{\texttt{ab}, \texttt{ba}\}$, $x_3x_4 \in \{\texttt{ab}, \texttt{ba}\}$ e $y \in \{\texttt{ab}, \texttt{ba}\}^\star$. Concluimos 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 $i \geq 0$.

Observe ainda que a condição do lema **nada diz sobre as palavras de L de comprimento menor que quatro.** *Em particular, é claramente falso que* $\texttt{b}$*, que é uma palavra de L, tenha alguma subpalavra diferente de ε que possa ser repetida o número de vezes que se quiser.*

5.2.2 Aplicações do Lema da Repetição

Para mostrar que uma dada linguagem L não é regular, podemos tentar mostrar que não satisfaz a condição do Lema da Repetição.
Para isso, precisamos de mostrar que **qualquer que seja o natural** n que se considere, **existem palavras** x **em** L que são suficientemente grandes (ou seja, **cujo comprimento não é inferior a** n), não tendo tais palavras qualquer subpalavra não vazia, que possa ser retirada ou repetida o número de vezes que se quiser. Ou seja, qualquer que seja a subpalavra v de x que se considere, ou não pode ser retirada ou existe pelo menos um i tal que a palavra que obtinhamos por repetição i vezes de v em x não era de L.
O lema diz que se essa palavra v existisse, então poderia ser encontrada num prefixo de x de comprimento não superior a n. Por isso, temos que arranjar uma forma de analisar **todos os prefixos de** x **de tamanho até** n.
Formalmente, basta mostrar que a linguagem satizfaz a negação da condição do Lema da repetição. Isto é

$$L \text{ regular} \Rightarrow \text{condição Lema Repetição}$$

$$\equiv$$

$$\neg\text{condição Lema Repetição} \Rightarrow \neg L \text{ regular}$$

Isto é, se L verifica a negação da condição do Lema da Repetição então L não é regular
Essa negação é:

$$\neg(\exists n > 0 \forall x \in L\ (|x| \geq n \Rightarrow \exists u, v, w (x = uvw, |uv| \leq n, v \neq \varepsilon) \wedge \forall i \geq 0\ uv^i w \in L))$$

$$\leftrightarrow$$

$$\forall n > 0 \exists x \in L\ (|x| \geq n \wedge \forall u, v, w (x = uvw, |uv| \leq n, v \neq \varepsilon) \Rightarrow \exists i \geq 0\ uv^i w \notin L)$$

Exemplo 5.10. *Vamos mostrar que* $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$ *não é regular, mostrando que* L *não satisfaz a condição do Lema da Repetição.*

Seja $n \in \mathbb{N} \setminus \{0\}$ *qualquer. Escolhemos* $x = \mathtt{0}^n\mathtt{1}^n$. *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* $|uv| \leq n$, *no prefixo* uv *só existem* `0`*'s pois* $|uv| \leq n$. *Sendo* $|v| \geq 1$, $v \neq \varepsilon$, *e por isso se "cortarmos"* v, *alteramos o número de* `0`*'s sem alterar o número de* `1`*'s. "Cortar" corresponde a tomar* $i = 0$. *Se considerarmos* $i = 0$, *vem* $uv^0w = uw$. *Tem-se* $uw \notin L$ *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.*

Note que não recorremos explicitamente a autómatos, como fizemos nos Exemplos **??** *e* **??**.

Exemplo 5.11. *A linguagem* $L = \{\mathtt{0}^p \mid p \text{ primo}\}$ *não satisfaz o Lema da Repetição.*

De facto, seja $n \in \mathbb{N} \setminus \{0\}$ *qualquer. 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,* $|\ uv^iw\ | = |u| + i\,|v| + |w|$ *e como* $|uv| \leq n$ *implica* $|v| \leq n$, *conclui-se que* $|uw| = M - |v| > n$. *Então, se se escolher* $i = |u| + |w|$, *vem*

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

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* $uv^iw \notin L$.
Em conclusão, L *não satisfaz a condição do Lema da Repetição e por isso não é regular.*

Exercício 5.3.3. Com base na linguagem do Exemplo **??**, mostre que qualquer que seja o alfabeto Σ, existem linguagens de alfabeto Σ não regulares.

Se se conseguir mostrar que uma dada linguagem não satisfaz a condição do lema da repetição, pode concluir-se que a linguagem não é regular. Mas, se a linguagem satisfizer a condição do lema, nada se pode dizer, pois existem linguagens que satisfazem essa condição e não são regulares, como se ilustra no Exemplo **??**.

Exemplo 5.12. *A linguagem* $L = \{0^k \mid k \in \mathbb{N}\} \cup \{1^r 0^{p^2} \mid r, p \in \mathbb{N} \setminus \{0\}\}$ *de alfabeto* $\Sigma = \{0, 1\}$, **satisfaz a condição do lema da repetição (para lings. regulares) para** $n = 1$**.**
De facto, se $x \in L$ *e* $|x| \geq 1$ *então* x *está num dos dois casos seguintes:*

- $x = 00^k$ *para algum* $k \geq 0$*, ou*
- $x = 11^s 0^{q^2}$ *para algum* $s \geq 0$ *e algum* $q \geq 1$*.*

Se for $x = 00^k$ *então escolhemos* $u = \varepsilon$*,* $v = 0$*, e* $w = 0^k$*, e sendo* $uv^i w = 0^{k+i}$*, concluimos que* $\forall i \in \mathbb{N}\ uv^i w \in L$*.*
Se for $x = 11^s 0^{q^2}$ *então escolhemos* $u = \varepsilon$*,* $v = 1$*, e* $w = 1^s 0^{q^2}$*, e concluimos que se* $i \geq 1$ *ou* $s \geq 1$ *então* $uv^i w \in \{1^r 0^{p^2} \mid r, p \in \mathbb{N} \setminus \{0\}\}$*, e se* $i = s = 0$ *então* $uv^i w \in \{0\}^\star$*.*
Portanto, para $n = 1$ *tem-se*

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

Mas L **não é regular.** *De facto, suponhamos que* L *era regular e que existia um autómato finito determinístico que aceitava* L*. Como o autómato é finito, tem um conjunto de estados finito, e consequentemente o conjunto de estados finais também é finito.*
Assim, existem inteiros positivos p *e* q *distintos tais que* 10^{p^2} *e* 10^{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* $10^{p^2}0^{(p+1)^2-p^2}$ *e* $10^{q^2}0^{(p+1)^2-p^2}$ *deveriam levar o autómato ao mesmo estado. Mas,* $10^{p^2}0^{(p+1)^2-p^2} = 10^{(p+1)^2}$ *que é da linguagem, e* $10^{q^2}0^{(p+1)^2-p^2} = 10^{q^2+2p+1}$ *não é da linguagem. 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* 10^{q^2+2p+1} *é aceite e não é da linguagem, ou* $10^{(p+1)^2}$ *não é aceite e é da linguagem. Tal é absurdo. Logo o autómato que supusemos que existia não existe.*

Depois de estudar o exemplo **??**, não deve ter dificuldade em resolver o problema seguinte.

Exercício 5.3.4. Mostre que a linguagem das palavras de alfabeto $\{0\}$ cujo comprimento é um quadrado perfeito não é regular.

5.2.3 Jogos entre a Alice e o Bob

A prova da negação da condição do lema da repetição pode ser vista como um jogo entre dois jogadores, a Alice e o Bob, tal que a Alice (que representa o quantificador existêncial) quer provar que L não é regular e o Bob quer provar que L é regular. Recordemos:

$$\forall n > 0 \exists x \in L\ (|x| \geq n \wedge \forall u, v, w (x = uvw, |uv| \leq n, v \neq \varepsilon) \Rightarrow \exists i \geq 0\ uv^i w \notin L)$$

O jogo consiste em:

1. Alice ($\exists$) selecciona a linguagem L que quer provar não regular
2. Bob ($\forall$) escolhe um n
3. Alice ($\exists$) escolhe $x \in L$ que depende do n e tal que $|x| \geq n$
4. Bob ($\forall$) escolhe u, v, w tal que $x = uvw$, $|uv| \leq n$ e $v \neq \varepsilon$
5. Alice ($\exists$) escolhe $i \geq 0$

A Alice ganha se $uv^i w \notin L$ e Bob ganha se tal i não existir. Se a Alice ganhar podemos concluir que L não é regular.
Usando este método consideremos os exemplos anteriores.

Exemplo 5.13. *Mostar que* $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$ *não satisfaz o Lema da Repetição.*

1. *Alice quer provar que L não é regular.*
2. *Bob toma um* $n > 0$.
3. *Alice escolhe* $x = \mathtt{0}^n\mathtt{1}^n$. *Tem-se que* $x \in L$ *e* $|x| = 2n > n$.
4. *Bob toma u, v e w tais que* $x = uvw$ *com* $|v| \geq 1$ *e* $|uv| \leq n$. *Em uv só existem* 0*'s e v tem pelo menos um* 0. *(Porquê?)*
5. *Alice escolhe* $i = 0$.

Vem que $uv^0w = uw$ *e* $uw \notin L$. *(Porquê?) Logo a Alice ganha, isto é, L não satisfaz a condição do Lema da Repetição, e por isso não é regular.*

Exemplo 5.14. *A linguagem* $L = \{\mathtt{0}^p \mid p \text{ primo}\}$ *não satisfaz o Lema da Repetição*

1. *Alice quer provar que L não é regular.*
2. *Bob toma um* $n > 0$.
3. *Alice escolhe* $x = \mathtt{0}^M$ *com* $M > 2n$ *e M primo. Tem-se* $x \in L$ *e* $|x| = M > 2n > n$.
4. *Bob toma u,v e w com* $x = uvw$, $|v| \geq 1$ *e* $|uv| \leq n$

5. *Alice tem de encontrar $i \geq 0$ tal que $uv^iw \notin L$, isto é, tal que $|uv^iw|$ não seja primo. Mas, $| uv^iw | = |u| + i\,|v| + |w|$. Alice escolhe $i = |u| + |w|$.*

   Basta ver que $|u| + (|u| + |w|)|v| + |w|$ não é primo!

   Como $|uv| \leq n$ implica $|v| \leq n$, vem $|uw| = |u| + |w| = M - |v| > n$.

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

   com $1 + |v| \geq 2$ e $|u| + |w| \geq 2$. Consequentemente, $(|u| + |w|)(1 + |v|)$ não é primo . Logo, para $i = |u| + |w|$, tem-se $uv^iw \notin L$. Alice ganha.q

5.2.4 Uso de propriedades de fecho

Para provar que uma linguagem não é regular também podemos invocar as propriedades de fecho das linguagens regulares. Este método permite reduzir uma linguagem complicada a uma mais simples que já sabemos que não é regular, e assim evitar o uso do lema da repetição.

Exemplo 5.15. *Seja*

$$L = \{x \in \{\mathtt{0},\mathtt{1}\}^\star |\ \textit{o número de } \mathtt{0}\textit{'s em } x \textit{ é igual ao número de } \mathtt{1}\textit{'s}\}$$

Para mostrar que L não é regular, suponhamos por contradição que L é regular.
Então $L \cap L(0^\star 1^\star)$ também é regular!
Mas

$$L \cap L(0^\star 1^\star) = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$$

a qual já mostramos que não era regular.
`Absurdo!` O absurdo resulou de supor que L era regular. Logo L não é regular.

Exercício 5.3.5. Para cada uma das seguintes linguagens, diga, justificando, se ela é ou não regular. Use o lema da repetição (para provar que uma linguagem *não é* regular) e/ou as propriedades de fecho para $\cup$, $\cap$, $\setminus$ (complementar), $\cdot$ (concatenação) e $*$ (fecho de Kleene).

a) $\{\mathtt{1}^n : n$ é par ou múltiplo de três$\}$

b) $\{\mathtt{1}^n : n$ é um primo inferior a 1000$\}$

c) $\{\mathtt{0}^i\mathtt{1}^j : i \neq j\}$

d) $\{a^nb^mc^k \mid n \geq 0,\ m \geq 0,\ k \geq 0,\ \}$

e) $\{a^nb^nc^n \mid n \leq 100\}$

f) $\{a^nb^mc^k \mid n \geq 0,\ m \geq 0,\ k = m + n\}$

g) $\{a^n b^n c^n \mid n \geq 0\}$

h) $\{a^n b^m \mid m \geq 0,\ n = 2m\}$

i) $\{ww^R : w \in \{0,1\}^*\}$ (w^R é a palavra w invertida)

j) $\{ww : w \in \{0,1\}^*\}$

5.3 Problemas decídiveis para Linguagens Regulares

Dada uma classe de linguages é importante saber se existem algoritmos (e qual a sua eficicia) para responder a perguntas como:

- Uma dada linguagem é vazia?
- Dada uma palavra w ela pertence a uma dada linguagem?
- Dadas as descrições de duas linguagens elas correspondem à mesma linguagem?

No caso das linguagens regulares, existem algoritmos para responder às seguintes questões:

1. Dado um autómato finito $\mathcal{A}$ e $x \in \Sigma^\star$, é verdade que $x \in L(\mathcal{A})$?
2. Dado um autómato finito $\mathcal{A}$, é $L(\mathcal{A}) = \emptyset$, i.e, $\mathcal{A}$ não aceita nenhuma palavra ?
3. Dado um autómato finito $\mathcal{A}$, é $L(\mathcal{A}) = \Sigma^\star$, i.e, $\mathcal{A}$ aceita todas as palavras sobre Σ ?
4. Dados dois autómatos finitos $\mathcal{A}_1$ e $\mathcal{A}_2$, é verdade $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$?
5. Dados dois autómatos finitos $\mathcal{A}_1$ e $\mathcal{A}_2$, é verdade $L(\mathcal{A}_1) = L(\mathcal{A}_2)$?

Ou, anlogamente, podem-se exprimir os problemas anteriores em termos de expressões regulares.

5.3.1 Testar se uma Palavra Pertence a uma Linguagem Regular

Dada uma palavra x e uma linguagem regular L, será que $x \in L$? Suponhamos que L é representada por um autómato finito deterministico $\mathcal{A}$. Podemos então ter o seguinte algoritmo:

1. Simular o processamento de $\mathcal{A}$ com x, começando no estado inicial (isto é calcular $\widehat{\delta}(q_0, x)$).
2. Se $\mathcal{A}$ terminar de processar x num estado final, então a resposta é *sim*, senão é *não*.

O tempo de execução é $O(n)$ em que $n = |x|$.

5.3.2 Testar se uma Linguagem é Vazia

Claro que se uma linguagem regular L for representada pela expressão regular $\emptyset$ o problema resolve-se imediatamente. Mas a expressão regular correspondente pode não ter aquela forma (lembrar a sua construção a partir do autómato...) e mesmo assim alinguagem ser vazia. Seja $\mathcal{A}$ um autómato que aceite L. O autómato $\mathcal{A}$ não aceita nenhuma palavra, se não houver nenhum caminho entre o estado inicial e algum estado final. Portanto o problema corresponde a encontrar caminhos num grafo entre o estado inicial e algum estado final. Podemos considera o seguinte m algoritmo recursivo:

1. Determinar o conjunto P dos estados atingíveis do estado inicial:

   Base. O estado inicial é atingível do estado inicial

   Indução. Se o estado s é atingível do estado inicial e existe um arco de s para s', então s' também é atingível.

2. Se algum estado final pertencer a P , a resposta é *não* senão é *sim*.

O tempo de execução é $O(n^2)$ em que n é o número de estados.

Exercício 5.4. *Se L for descrita por uma expressão regular, descrever um algoritmo para decidir se L é vazia.* $\diamond$

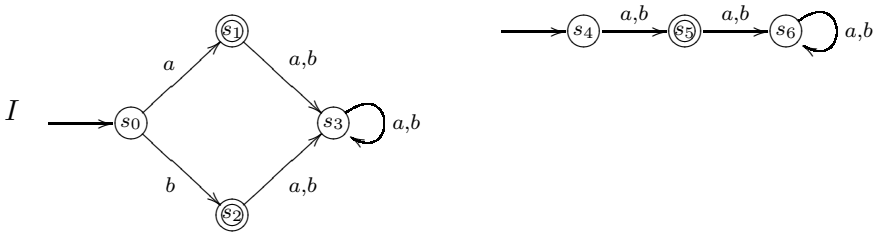
Exercício 5.5. *Encontrar um algoritmo para decidir se $L(\mathcal{A}) = \Sigma^\star$.* $\diamond$

5.4 Equivalência e Minimização de Autómatos

Dados dois autómatos $\mathcal{A}_1$ e $\mathcal{A}_2$, pretende-se saber se descrevem a mesma linguagem, isto é se são equivalentes ($L(\mathcal{A}_1) = L(\mathcal{A}_2)$). Para tal, vamos ver que dado qualquer AFD existe um autómato equivalente que tem um número mínimo de estados e que é *único* a menos renomeação dos estados.
Assim para responder à pergunta $L(\mathcal{A}_1) = L(\mathcal{A}_2)$, bastará minimizar $\mathcal{A}_1$ e $\mathcal{A}_2$ e ver se os autómatos minimos resultantes são iguais (a menos renomeação de estados).

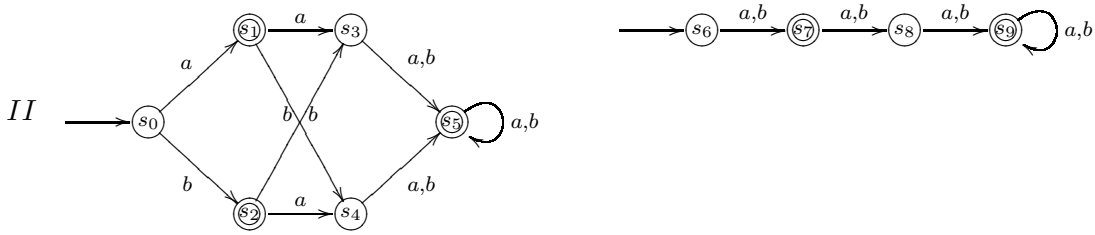
5.4.1 Minimização de autómatos finitos

Exemplo 5.16. *Consideremos os seguintes autómatos*



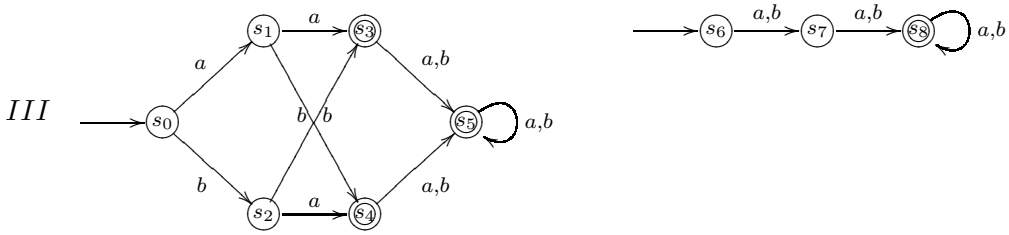
Ambos os autómatos aceitam a linguagem $\{a, b\}$. Os estados s_1 e s_2 podem colapsar no estado s_5.

Exemplo 5.17. *Consideremos os seguintes autómatos*



Ambos os autómatos aceitam a linguagem $\{a, b\} \cup \{x \in \{a, b\}^\star || x| \geq 3\}$. Os estados s_3 e s_4 são equivalentes*, pois ambos conduzem a s_5. Depois de colapsados (em s_8), s_1 e s_2 também se podem colapsar (em s_7).*

Exemplo 5.18.



Ambos os autómatos aceitam a linguagem $\{x \in \{a, b\}^\star || x| \geq 2\}$. Os estados s_3, s_4 e s_5 podem ser colapsados num só (em s_8).

Temos que começar por determinar quando é que dois estados são equivalentes, i.e.:

- Como determinar se dois estados podem colapsar?
- Como colapsá-los?
- Como garantir que não se podem colapsar mais estados?

Seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um autómato finito determinístico (sem estados inacessíveis do estado inicial):

- Se $s \in F$ e $s' \notin F$, s e s' não se podem colapsar, dado que depois não se podia decidir se o estado resultante estava ou não em F.
- Se s e s' forem colapsados, então $\delta(s, a)$ e $\delta(s', a)$ também devem ser colapsados. Caso contrário, não se mantinha o determinismo.

Definição 5.1. *Dois estados s e s' dizem se* equivalentes *se para todo o $x \in \Sigma^\star$, $\widehat{\delta}(s, x) \in F$ se e só se $\widehat{\delta}(s', x) \in F$. Caso contrário, dizem-se* distinguíveis.

Dois estados equivalentes podem ser colapsados e dois distinguíveis não podem ser colapsados. Formalmente,considere-se a relação em $\approx \subseteq S \times S$ definida por:

$$s \approx s' \iff \forall x \in \Sigma^\star (\widehat{\delta}(s, x) \in F \leftrightarrow \widehat{\delta}(s', x) \in F) \tag{5.1}$$

É fácil ver que a relação $\approx$ é de equivalência:

reflexiva: $s \approx s$ para todo o s;

simétrica: se $s \approx s'$ então é óbvio $s \approx s'$;

transitiva se $s \approx s'$ e se $s' \approx s''$ então também $s \approx s''$; (verificar!)

Então, $\approx$ divide S em classes de equivalência (subconjuntos de S dois a dois disjuntos e cuja intersecção é S):

$$[s] = \{s' | s \approx s'\}$$

Podemos então definir o *autómato quociente* $\mathcal{A}/\approx$, onde os estados vão ser as classes de equivalência de $\approx$. Este autómato será o autómato mínimo equivalente a $\mathcal{A}$.
Seja $\mathcal{A}/\approx = (S', \Sigma, \delta', {s_0}', F')$, onde:

$$\begin{aligned} S' &\stackrel{def}{=} \{[s] | s \in S\} \\ \delta'([s], a) &\stackrel{def}{=} [\delta(s, a)] \\ s_0' &\stackrel{def}{=} [s_0] \\ F' &\stackrel{def}{=} \{[s] | s \in F\} \end{aligned}$$

Temos que verificar que δ' está bem definido e que $L(\mathcal{A}) = L(\mathcal{A}/\approx)$.

Lema 5.2. *Se $s \approx s'$, então $\delta(s, a) \approx \delta(s', a)$.*

Dem: Suponhamos que $s \approx s'$. Seja $x \in \Sigma^\star$ e $a \in \Sigma$.

$$\begin{aligned} \widehat{\delta}(\delta(s,a),x) \in F &\iff \widehat{\delta}(s,ax) \in F \\ &\iff \widehat{\delta}(s',ax) \in F \\ &\iff \widehat{\delta}(\delta(s',a),x) \in F \end{aligned}$$

□

Então, δ' está bem definido: se $[s] = [s']$, então $\delta'([s]) = [\delta(s,a)] = [\delta(s',a)] = \delta'[s']$.

Lema 5.3. $s \in F \iff [s] \in F'$.

Dem: $\Rightarrow$ pela definição de F'.
$\Leftarrow$ É necessário que se $s \approx s'$ e $s \in F$ então $s' \in F$. Mas basta tomar $x = \epsilon$ na definição de $\approx$ para tal ser verdade. □

Lema 5.4. $\forall x \in \Sigma^\star \forall s \in S,\ \widehat{\delta'}([s],x) = [\widehat{\delta}(s,x)]$

Dem: Por indução em $|x|$. Se $x = \epsilon$, $\widehat{\delta'}([s],\epsilon) = [s]$ e $[\widehat{\delta}(s,\epsilon)] = [s]$. Suponhamos que $\widehat{\delta'}([s],x) = [\widehat{\delta}(s,x)]$ e seja $a \in \Sigma$

$$\begin{aligned} \widehat{\delta'}([s],xa) &= \delta'(\widehat{\delta'}([s],x),a) \\ &= \delta'([\widehat{\delta}(s,x)],a) \\ &= [\delta(\widehat{\delta}(s,x),a)] \\ &= [\widehat{\delta}(s,xa)] \end{aligned}$$

□

Proposição 5.4. $L(\mathcal{A}) = L(\mathcal{A}/\approx)$

$$\begin{aligned} x \in L(\mathcal{A}/\approx) &\iff \widehat{\delta'}([s_0],x) \in F' \\ &\iff [\widehat{\delta}(s_0,x)] \in F' \\ &\iff \widehat{\delta}(s_0,x) \in F \\ &\iff x \in L(\mathcal{A}) \end{aligned}$$

Para ver que não se pode colapsar mais, basta verificar que se construirmos o quociente do autómato quociente obtemos o mesmo autómato.

5.4.1.1 Algoritmo para determinar $\approx$

Supondo um AFD sem estados inacessíveis, o algoritmo marca todos os pares de estados (s, s') que são distinguíveis.

1. Construir uma tabela com todos os pares (s, s') sem estarem marcados.
2. Marcar todos os pares de estados (s, s') tal que $s \in F$ e $s' \in S \setminus F$
3. Repetir até não haver alteração: Se existe um par não marcado (s, s') tal que $(\delta(s, a), \delta(s', a))$ está marcado para algum $a \in \Sigma$, então marcar (s, s').

Proposição 5.5. *Um par (s, s') é marcado pelo algoritmo se e só se s e s' são distinguíveis, i.e., $s \not\approx s'$.*

Dem: $\Rightarrow$ Suponhamos que (s, s') está marcado. Mostramos por indução número de passos do algoritmo (que sabemos ser finito) que se (s, s') é marcado no passo n então $s \not\approx s'$.

Base. **Um passo** Se (s, s') foi marcado no passo **??**, então ou $s \in F$ e $s' \notin F$, ou vice-versa. Em ambos os casos, para $x = \epsilon$ obtemos $s \not\approx s'$.

Indução. Se (s, s') foi marcado no passo **??**, então para algum $a \in \Sigma$, o par $(\delta(s, a), \delta(s', a))$ já estava marcado anteriormente. Então, por hipótese de indução, $\delta(s, a) \not\approx \delta(s', a)$ e, pelo Lema **??**, $s \not\approx s'$.

$\Leftarrow$ Suponhamos que $s \not\approx s'$. Por definição, existe $x \in \Sigma^\star$ tal que $\widehat{\delta}(s, x) \in F$ e $\widehat{\delta}(s', x) \notin F$, ou vice-versa. Mostramos por indução em $|x|$.

Base. Se $x = \epsilon$, então ou $s \in F$ e $s' \notin F$ ou vice-versa. Logo o par (s, s') é marcado no passo **??**.

Indução. Se $x = ay$, então $\widehat{\delta}(\delta(s, a), y) \in F$ e $\widehat{\delta}(\delta(s', a), y) \notin F$ ou vice-versa. Então, $\delta(s, a) \not\approx \delta(s', a)$ e, por hipótese de indução (porque $|y| < |x|$), $(\delta(s, a), \delta(s', a))$ foi marcado pelo algoritmo e então (s, s') será marcado na passagem seguinte.

□

Conclusão:
Dado um autómato finito determínstico que aceita uma linguagem L existe um único autómato finito determínstico que aceita L e cujo número de estados é mínimo.

Exemplo 5.19. *Minimizar o primeiro autómato do grupo* **??***II, cuja função de transição é:*

δ	a	b
$\rightarrow s_0$	s_1	s_2
$\star s_1$	s_3	s_4
$\star s_2$	s_4	s_3
s_3	s_5	s_5
s_4	s_5	s_5
$\star s_5$	s_5	s_5

Vamos construir uma tabela com os pares $(s, s') \in S \times S$*. Como a tabela é simétrica e os pares* (s, s) *nunca serão marcados, basta-nos considerar a parte triangular inferior da tabela. Após o passo* **??***, a tabela fica:*

0					
–	1				
–	–	2			
–	–	–	3		
–	–	–	–	4	
–	–	–	–	–	5

Após o passo **??***, a tabela fica:*

0					
x	1				
x	–	2			
–	x	x	3		
–	x	x	–	4	
x	–	–	x	x	5

Considerem-se os pares não marcados:

(s_0, s_3)*: por a obtemos o par* (s_1, s_5) *e por b o par* (s_2, s_5) *e nenhum deles marcado, portanto não se marca* (s_0, s_3)

(s_0, s_4)*: também não se marca*

(s_1, s_2)*: por a obtemos o par* (s_3, s_4) *e por b o par* (s_4, s_3) *e nenhum deles marcado, portanto não se marca* (s_1, s_2)

(s_1, s_5)*: por a obtemos o par* (s_3, s_5) *que está marcado, portanto marca-se* (s_1, s_5)

(s_2, s_5)*: por a obtemos o par* (s_4, s_5) *que está marcado, portanto marca-se* (s_2, s_5)

(s_3, s_4): por a e b obtemos o par (s_3, s_5) que nunca será marcado, portanto não se marca (s_3, s_4) A tabela fica:

$$
\begin{array}{cccccc}
0 & & & & & \\
x & 1 & & & & \\
x & - & 2 & & & \\
- & x & x & 3 & & \\
- & x & x & - & 4 & \\
x & \mathrm{x} & \mathrm{x} & x & x & 5
\end{array}
$$

Fazendo outra passagem pela tabela: agora marca-se (s_0, s_3) porque (s_1, s_5) agora está marcado; analogamente também se marca (s_0, s_4); os restantes ficam não se podem marcar

$$
\begin{array}{cccccc}
0 & & & & & \\
x & 1 & & & & \\
x & - & 2 & & & \\
\mathrm{x} & x & x & 3 & & \\
\mathrm{x} & x & x & - & 4 & \\
x & x & x & x & x & 5
\end{array}
$$

Fazendo outra passagem pela tabela:
os restantes pares continuam sem se marcar e portanto não são feitas novas marcas. O algoritmo termina.
Concluímos que $s_1 \approx s_2$ e $s_3 \approx s_4$ e que segundo autómato apresentado é o mínimo.

Capítulo 6

Gramáticas e Linguagens independentes de contexto

6.1 Gramáticas

Nesta secção vamos introduzir gramáticas formais para caracterização das linguagens, estudando fundamentalmente as *gramáticas independentes de contexto*. Começamos por apresentar alguns exemplos.
Vimos que a seguinte linguagem não é regular

$$L = \{0^n 1^n \mid n \geq 0\}$$

Contudo podemos fácilmente dar uma definição indutiva das suas palavras:

1. $\epsilon \in L$
2. Se $x \in L$ então $0x1 \in L$

L é a linguagem de $\{0,1\}^\star$ obtida por aplicação, uma ou mais vezes, da seguintes regras:

(1) $P \to \epsilon$

(2) $P \to 0P1$

Exemplo 6.1. *Seja L a linguagem de alfabeto $\{\mathtt{a},\mathtt{b}\}$ das palavras obtidas por aplicação,* uma ou mais vezes*, das regras seguintes.*

(r1) $\mathtt{aaa} \in L$
(r2) $\alpha\mathtt{bb}\beta \in L$, *quaisquer que sejam* $\alpha, \beta \in L$.

Podemos representar (i) e (ii) pelas duas regras seguintes

$$\begin{aligned} <Palavra> &\rightarrow \texttt{aaa} \\ <Palavra> &\rightarrow <Palavra>\texttt{bb}<Palavra> \end{aligned}$$

as quais definem a categoria gramatical $<Palavra>$*, dizendo que uma* $<Palavra>$ *é* `aaa` *ou a justaposição de três sequências — uma qualquer* $<Palavra>$*,* `bb` *e uma qualquer* $<Palavra>$*.*

Alternativamente, como $L = \mathcal{L}((\texttt{aaabb})^\star\texttt{aaa})$*, podemos ainda definir* L *como sendo a linguagem gerada por aplicação das regras:*

$$\begin{aligned} <Palavra> &\rightarrow \texttt{aaa} \\ <Palavra> &\rightarrow \texttt{aaabb}<Palavra> \end{aligned}$$

Exemplo 6.2. *Seja* $L_1 = \{\texttt{0}^{2k}\texttt{1}^k \mid k \in \mathbb{N}\}$*. A linguagem* L_1 *é o menor subconjunto de* $\{\texttt{0},\texttt{1}\}^\star$ *que satisfaz as duas condições (i) e (ii) seguintes.*

$$\begin{aligned} &(i) \quad \varepsilon \in L_1 \\ &(ii) \quad \forall x \in L_1 \quad \texttt{00}x\texttt{1} \in L_1 \end{aligned}$$

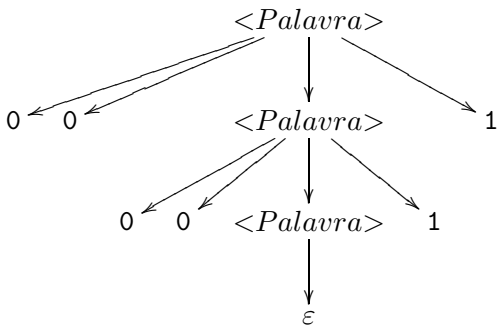
Equivalentemente, podemos dizer que L_1 *é constituída por todas as palavras que possam ser formadas por aplicação, uma ou mais vezes, das regras* (r_1) *e* (r_2) *seguintes*

$$\begin{aligned} &(r_1) \quad \varepsilon \in L_1 \\ &(r_2) \quad \textit{Qualquer que seja } x\textit{, se } x \in L_1 \textit{ então } \texttt{00}x\texttt{1} \in L_1 \end{aligned}$$

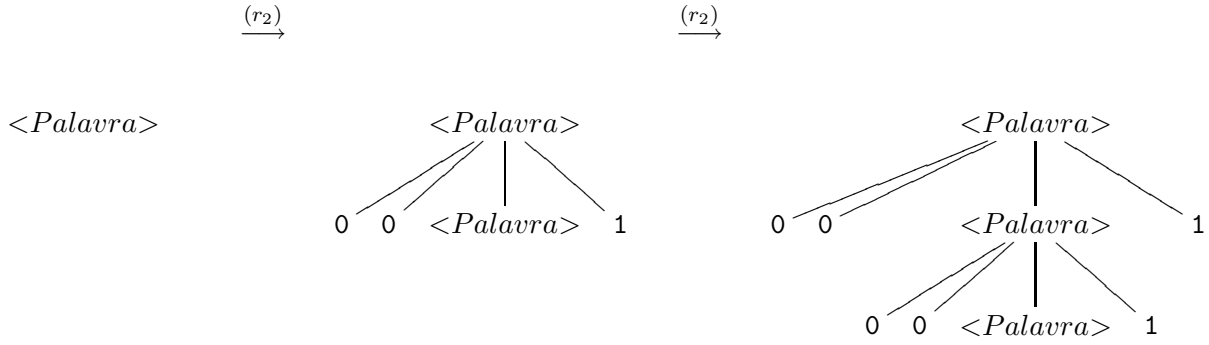
à semelhança do exemplo anterior, podemos definir as palavras de L_1 *como sendo da categoria* $<Palavra>$*, assim definida*

$$\begin{aligned} <Palavra> &\rightarrow \varepsilon \\ <Palavra> &\rightarrow \texttt{00}<Palavra>\texttt{1} \end{aligned}$$

A palavra `000011` *é uma* $<Palavra>$*, como se ilustra pelo seguinte esquema, a que se chama* árvore de derivação (ou árvore sintáctica) *para a palavra* `000011`*.*



a qual é assim construída (para simplificar representamos segmentos em vez de setas):



restando aplicar a regra $<Palavra> \rightarrow \varepsilon$.

Exemplo 6.3. *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* $\alpha_1, \alpha_2 \in F$

Considerando que as palavras da linguagem F são da categoria P, podemos escrever as regras seguintes:

$$P \rightarrow \texttt{x}$$
$$P \rightarrow \texttt{f(}P\texttt{,}P\texttt{)}$$

Alguns exemplos de palavras desta linguagem:

`f(x,x)`

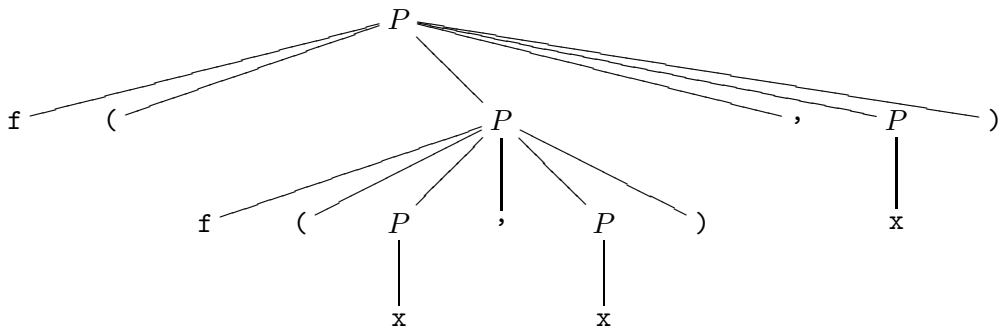
`f(f(x,x),x)`

`f(x,f(x,x))`

`f(f(x,x),f(x,x))`

`f(f(x,x),f(f(x,x),x))`

A árvore sintáctica de `f(f(x,x),x)`*:*



Exemplo 6.4. *Seja* $L_2 = \{0^{2k}1^k \mid k \geq 0\} \cup \{1^k0^{2k} \mid k \geq 0\}$*, a qual pode ser definida indutivamente pelas regras (i)—(iii).*

(i) $\varepsilon \in L_2$

(ii) *se* $x \in L_2$ *e* $x \notin \{0,1\}^\star\{1\}$ *então* $1x00 \in L_2$

(iii) *se* $x \in L_2$ *e* $x \notin \{0,1\}^\star\{0\}$ *então* $00x1 \in L_2$

Se observarmos L_2*, vemos que é constituída por dois tipos de palavras: as da forma* $0^{2k}1^k$*, que passamos a referir como sendo da categoria* A *e as da forma* 1^k0^{2k}*, que passamos a referir como sendo da categoria* B*. às palavras de* L_2*, as quais podem ser do tipo* A *ou do tipo* B*, atribuimos a categoria* S*. As regras da gramática são*

$$\begin{aligned} S &\to A \\ S &\to B \end{aligned}$$

$$\begin{aligned} A &\to \varepsilon \\ A &\to 00\,A\,1 \end{aligned}$$

$$\begin{aligned} B &\to \varepsilon \\ B &\to 1\,B\,00 \end{aligned}$$

Note que cada palavra de L_2*, com excepção da palavra* ε*, tem uma só árvore sintáctica.*
A palavra ε *tem duas árvores sintácticas:*

Por esta razão, a gramática diz-se ambígua*. Se as regras fossem as indicadas abaixo, então a gramática já não seria ambígua.*

$$\begin{aligned} S &\to \varepsilon \\ S &\to 00\,A\,1 \\ S &\to 1\,B\,00 \end{aligned}$$

$$\begin{aligned} A &\to \varepsilon \\ A &\to 00\,A\,1 \end{aligned}$$

$$\begin{aligned} B &\to \varepsilon \\ B &\to 1\,B\,00 \end{aligned}$$

Para cada palavra da linguagem existiria uma e uma só árvore sintáctica (a gramática fixaria uma estrutura sintáctica única para cada palavra).

Exemplo 6.5. *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 constituída pelas palavras da categoria* S *definida pelas regras seguintes.*

$$S \rightarrow \mathtt{0}\,T\,\mathtt{1}$$

$$\begin{aligned} T &\rightarrow \mathtt{0}\,T \\ T &\rightarrow \mathtt{1}\,T \\ T &\rightarrow \varepsilon \end{aligned}$$

Por vezes, vamos escrever as mesmas regras duma forma abreviada, como se segue:

$$\begin{aligned} S &\rightarrow \mathtt{0}\,T\,\mathtt{1} \\ T &\rightarrow \mathtt{0}\,T \mid \mathtt{1}\,T \mid \varepsilon \end{aligned}$$

A linguagem $\{\mathtt{0},\mathtt{1}\}^\star$ *é gerada pelas regras para* T*.*

Exemplo 6.6. *A linguagem* $L = \{\mathtt{0}x_1 \ldots \mathtt{0}x_k\mathtt{0} \mid k \geq 1, x_i \in \mathcal{L}(\ (\mathtt{11})^\star\mathtt{11}\ )$ *para* $1 \leq i \leq k\}$ *é definida pela gramática*

$$\begin{aligned} S &\rightarrow \mathtt{0}\,X\,R \\ R &\rightarrow \mathtt{0}\,X\,R \mid \mathtt{0} \\ X &\rightarrow \mathtt{11}\,X \mid \mathtt{11} \end{aligned}$$

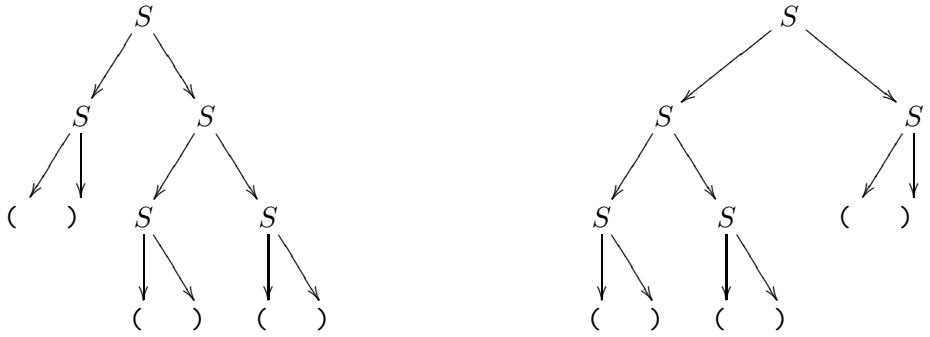
sendo constituída pelas palavras da categoria S*. Verificamos que*

- X *gera* $\mathcal{L}(\ (\mathtt{11})^\star\mathtt{11}\ )$*,*
- R *gera* $\mathcal{L}(\ (\mathtt{0}(\mathtt{11})^\star\mathtt{11})^\star\mathtt{0}\ )$*, e*
- S *gera* $L = \mathcal{L}(\ (\mathtt{0}(\mathtt{11})^\star\mathtt{11})^\star\mathtt{0}(\mathtt{11})^\star\mathtt{110}\ )$*.*

Exemplo 6.7. *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 duma 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 das 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*

$$\begin{array}{lcl} S & \rightarrow & X \mid XS \\ X & \rightarrow & (\,S\,) \mid (\,) \end{array}$$

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 6.8. *Vamos agora considerar uma gramática para expressões aritméticas.*

$$\begin{array}{lcllcllcl} E & \rightarrow & E{*}E & D & \rightarrow & \mathtt{0} & D & \rightarrow & \mathtt{6} \\ E & \rightarrow & E{+}E & D & \rightarrow & \mathtt{1} & D & \rightarrow & \mathtt{7} \\ E & \rightarrow & (E) & D & \rightarrow & \mathtt{2} & D & \rightarrow & \mathtt{8} \\ E & \rightarrow & I & D & \rightarrow & \mathtt{3} & D & \rightarrow & \mathtt{9} \\ I & \rightarrow & D & D & \rightarrow & \mathtt{4} & & & \\ I & \rightarrow & DI & D & \rightarrow & \mathtt{5} & & & \end{array}$$

Uma derivação para $\mathtt{5}*(\mathtt{6}+\mathtt{19})$

$$\begin{array}{rcl} E & \Rightarrow & E*E \Rightarrow I*E \Rightarrow D*E \Rightarrow 5*E \\ & \Rightarrow & 5*(E) \Rightarrow 5*(E+E) \Rightarrow 5*(I+E) \Rightarrow 5*(D+E) \\ & \Rightarrow & 5*(6+E) \Rightarrow 5*(6+I) \Rightarrow 5*(6+DI) \Rightarrow 5*(6+1I) \\ & \Rightarrow & 5*(6+1D) \Rightarrow 5*(6+19) \end{array}$$

Exemplo 6.9. *Que palavras de alfabeto* $\{(,),+,^\star,\emptyset,\boxed{\varepsilon},\mathtt{a},\mathtt{b}\}$ *são geradas por aplicação das regras seguintes?*

$$\begin{array}{lcl} S & \rightarrow & \mathtt{a} \mid \mathtt{b} \mid \emptyset \mid \boxed{\varepsilon} \\ S & \rightarrow & (S{+}S) \mid (SS) \mid (S^\star) \end{array}$$

Se admitirmos que $\boxed{\varepsilon}$ *está a representar o símbolo* ε*, podemos dizer que esta gramática descreve a linguagem das expressões regulares de alfabeto* $\{\mathtt{a}, \mathtt{b}\}$*. Saliente-se que se tivessemos escrito* $S \to \varepsilon$*, a gramática não geraria, por exemplo, a expressão* $(\mathtt{a} + \varepsilon)$*, mas geraria* $(\mathtt{a}+)$*, 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* $(\mathtt{a} + \boxed{\varepsilon})$*.*

Mais ainda, as palavras desta linguagem **são expressões** *regulares;* **não são linguagens** *regulares.*

Exemplo 6.10. *Este exemplo sugere uma motivação para o estudo das gramáticas no âmbito do curso — a descrição de linguagens de programação.*
Que palavras de $\{(,), :, >, <, =, +, -, *, /, 0, \ldots, 9, \mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}, \cdot, ;\}^\star$ *são geradas a partir de* S *por aplicação das regras seguintes?*

$$
\begin{array}{lcl}
S & \to & \mathtt{if} \cdot (C) \cdot \mathtt{then} \cdot \mathtt{goto} \cdot N \\
S & \to & \mathtt{goto} \cdot N \\
S & \to & V \mathtt{:=} E \\
S & \to & \mathtt{stop} \\
E & \to & (E\,O\,E) \mid (-E) \mid V \mid N \\
O & \to & + \mid - \mid * \mid / \\
C & \to & V\,X\,E \\
X & \to & \mathtt{>} \mid \mathtt{<} \mid \mathtt{>=} \mid \mathtt{<=} \mid \mathtt{=} \mid \mathtt{<>} \\
N & \to & D \mid D\,N \\
D & \to & \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \mathtt{3} \mid \mathtt{4} \mid \mathtt{5} \mid \mathtt{6} \mid \mathtt{7} \mid \mathtt{8} \mid \mathtt{9} \\
V & \to & L \mid L\,N \mid L\,V \\
L & \to & \mathtt{a} \mid \mathtt{b} \mid \mathtt{c} \mid \mathtt{d} \mid \mathtt{e} \mid \mathtt{f} \mid \mathtt{g} \mid \mathtt{h}
\end{array}
$$

e de P *?*

$$P \to N \cdot S \mid S \,;\cdot\cdot\, P$$

Exemplo duma palavra da linguagem gerada a partir de P*:*

$$\mathtt{2} \cdot \mathtt{a:=10};\cdot\cdot\, \mathtt{3} \cdot \mathtt{b:=35};\cdot\cdot\, \mathtt{4} \cdot \mathtt{if} \cdot (\mathtt{b} < \mathtt{a}) \cdot \mathtt{then} \cdot \mathtt{goto} \cdot \mathtt{8};\cdot\cdot\, \mathtt{6} \cdot \mathtt{b:=}(\mathtt{b} - \mathtt{a});\cdot\cdot\, \mathtt{7} \cdot \mathtt{goto} \cdot \mathtt{4};\cdot\cdot\, \mathtt{8} \cdot \mathtt{stop}$$

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 6.11. *A gramática seguinte permite a geração de um pequeno fragmento de frases em língua portuguesa.*

$$
\begin{array}{lcl}
\langle frase\rangle & \rightarrow & \langle frase\_nominal\rangle\langle frase\_verbal\rangle \\
\langle frase\_nominal\rangle & \rightarrow & \langle determinante\rangle\langle frase\_nominal1\rangle \\
\langle frase\_nominal1\rangle & \rightarrow & \langle nome\_c\rangle\langle adjs\rangle \\
\langle frase\_nominal1\rangle & \rightarrow & \langle nome\_p\rangle \\
\langle adjs\rangle & \rightarrow & \langle adj\rangle\langle adjs\rangle \\
\langle adjs\rangle & \rightarrow & \epsilon \\
\langle frase\_verbal\rangle & \rightarrow & \langle verbo\_i\rangle \\
\langle frase\_verbal\rangle & \rightarrow & \langle verbo\_t\rangle\langle frase\_nominal\rangle \\
\langle nome\_c\rangle & \rightarrow & \texttt{gato} \\
\langle nome\_c\rangle & \rightarrow & \texttt{rato} \\
\langle nome\_p\rangle & \rightarrow & \texttt{joana} \\
\langle verbo\_i\rangle & \rightarrow & \texttt{dorme} \\
\langle verbo\_t\rangle & \rightarrow & \texttt{comeu} \\
\langle determinante\rangle & \rightarrow & \texttt{o} \\
\langle adj\rangle & \rightarrow & \texttt{preto}
\end{array}
$$

Esta gramática deriva *a frase* `o gato preto comeu o rato`*, mas não a frase* `gato rato comeu o`*.*

$$
\begin{array}{rcl}
\langle frase\rangle & \Rightarrow & \langle frase\_nominal\rangle\langle frase\_verbal\rangle \\
& \Rightarrow & \langle determinante\rangle\langle frase\_nominal1\rangle\langle frase\_verbal\rangle \\
& \Rightarrow & \texttt{o}\ \langle frase\_nominal1\rangle\langle frase\_verbal\rangle \\
& \Rightarrow & \texttt{o}\ \langle nome\_c\rangle\langle adjs\rangle\langle frase\_verbal\rangle \\
& \Rightarrow & \texttt{o gato}\ \langle adjs\rangle\langle frase\_verbal\rangle \\
& \Rightarrow & \texttt{o gato}\ \langle adj\rangle\langle adjs\rangle\langle frase\_verbal\rangle \\
& \Rightarrow & \texttt{o gato preto}\ \langle frase\_verbal\rangle \\
& \Rightarrow & \texttt{o gato preto}\ \langle verbo\_t\rangle\langle frase\_nominal\rangle \\
& \Rightarrow & \texttt{o gato preto comeu}\ \langle frase\_nominal\rangle \\
& \Rightarrow & \texttt{o gato preto comeu}\ \langle determinante\rangle\langle frase\_nominal1\rangle \\
& \Rightarrow & \texttt{o gato preto comeu o}\ \langle frase\_nominal1\rangle \\
& \Rightarrow & \texttt{o gato preto comeu o}\ \langle nome\_c\rangle \\
& \Rightarrow & \texttt{o gato preto comeu o rato}
\end{array}
$$

6.2 Gramáticas Independentes de Contexto e Linguagens

Definição 6.1. *Uma* gramática independente de contexto *é um quádruplo*

$$\mathcal{G} = (V, \Sigma, P, S)$$

- V *é um conjunto finito não vazio de* **variáveis** *ou* **não terminais** *(ou categorias sintáticas)*
- Σ *é um conjunto finito não vazio, disjunto de* V*,* **alfabeto** *ou conjunto de* **terminais**
- $S \in V$ *é o* **símbolo inicial** *de* $\mathcal{G}$
- P *é uma relação binária finita de* V *em* $(V \cup \Sigma)^\star$*. Os elementos de* P *denominam-se* **produções** *ou* **regras**. *Se* $(X, w) \in P$ *escreve-se* $X \to w$

  Se existirem várias regras para um mesmo não terminal X *podemos abreviar numa, separando-as com* $|$ *do lado direito*

  $X \to \alpha_1, X \to \alpha_2, X \to \alpha_3$ *fica* $X \to \alpha_1|\alpha_2|\alpha_3$

Porquê *Independentes de contexto*? Porque as regras que definem um não terminal X não estão restritas a um contexto. Para gramáticas mais gerais (**Tipo 0**) podemos ter regras do tipo

$$\alpha \to \beta \quad \text{com} \quad \alpha, \beta \in (V \cup \Sigma)^\star, \ \ \alpha \neq \varepsilon$$

e no caso das GIC tem de se ter $\alpha \in V$! Mas não vamos considerar gramáticas desses tipos. As GICs permitem descrever conjuntos infinitos de um modo finito. São especialmente utilizadas para a descrição da sintaxe de linguagens programação.

Exemplo 6.12. *Considere as gramáticas independentes de contexto assim definidas.*

$$\begin{aligned}
\mathcal{G}_1 &= (\{T\}, \{\mathtt{0},\mathtt{1}\}, \{T \to \mathtt{0}T,\ T \to \mathtt{1}T,\ T \to \varepsilon\}, T)\\
\mathcal{G}_2 &= (\{Z,U\}, \{\mathtt{0},\mathtt{1}\}, \{Z \to \mathtt{0}U,\ U \to \mathtt{1}Z,\ U \to \varepsilon\}, Z)\\
\mathcal{G}_3 &= (\{Z,U\}, \{\mathtt{0},\mathtt{1}\}, \{Z \to \mathtt{0}U,\ U \to \mathtt{1}Z,\ U \to \varepsilon\}, U)\\
\mathcal{G}_4 &= (\{U\}, \{\mathtt{0},\mathtt{1}\}, \{U \to \mathtt{10}U,\ U \to \varepsilon\}, U)\\
\mathcal{G}_5 &= (\{U,Z\}, \{\mathtt{0},\mathtt{1}\}, \{Z \to \mathtt{0}U,\ U \to \mathtt{10}U,\ U \to \varepsilon\}, Z)\\
\mathcal{G}_6 &= (\{S\}, \{\mathtt{a},\mathtt{b}\}, \{S \to \mathtt{aaa},\ S \to \mathtt{aaabb}S\}, S)\\
\mathcal{G}_7 &= (\{S,C\}, \{\mathtt{a},\mathtt{b}\}, \{S \to \mathtt{a}S\mathtt{b},\ S \to C,\ C \to \mathtt{b}C\mathtt{a},\ C \to \mathtt{b}\}, S)
\end{aligned}$$

Note que **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**. *As gramáticas dadas geram as linguagens seguintes.*

$$\begin{aligned}
\mathcal{L}(\mathcal{G}_1) &= \{\mathtt{0},\mathtt{1}\}^\star \\
\mathcal{L}(\mathcal{G}_2) &= \{\mathtt{0}\}\{\mathtt{10}\}^\star = \mathcal{L}(\mathcal{G}_5) \\
\mathcal{L}(\mathcal{G}_3) &= \{\mathtt{10}\}^\star = \mathcal{L}(\mathcal{G}_4) \\
\mathcal{L}(\mathcal{G}_6) &= \{\mathtt{aaabb}\}^\star\{\mathtt{aaa}\} \\
\mathcal{L}(\mathcal{G}_7) &= \{\mathtt{a}^n\mathtt{b}^{m+1}\mathtt{a}^m\mathtt{b}^n \mid m \in \mathbb{N}, n \in \mathbb{N}\}
\end{aligned}$$

A **linguagem** que uma gramática gera é constituída pelas palavras que podem ser derivadas a partir do símbolo inicial (usando as regras). Para formalizar estação começamos por definir a noção de textbfderivação.

Exemplo 6.13. *Voltemos a considerar a gramática dada no Exemplo* **??**, *que agora definimos, mais formalmente, por*

$$\mathcal{G} = (\{S, R, X\}, \{\mathtt{0},\mathtt{1}\}, \mathcal{P}, S)$$

em que o conjunto das regras $\mathcal{P}$ *é consituído por*

$$S \;\to\; \mathtt{0}\,X\,R$$

$$R \;\to\; \mathtt{0}\,X\,R \;\mid\; \mathtt{0}$$

$$X \;\to\; \mathtt{11}\,X \;\mid\; \mathtt{11}$$

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 *substitui-se uma variável qualquer aplicando uma* regra. *Por exemplo:*

$$S \Rightarrow_{\mathcal{G}} \mathtt{0}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{011}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{011}\,X\,\mathtt{0} \Rightarrow_{\mathcal{G}} \mathtt{011110}$$

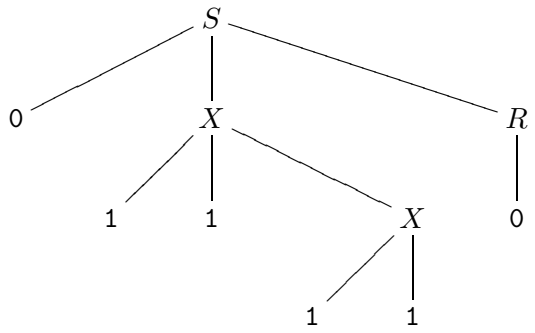
Existem outras derivações para **011110**. *Se substituirmos sempre a* variável *mais à esquerda, temos* **uma derivação** *que dizemos ser* **pela esquerda**.

$$S \Rightarrow_{\mathcal{G}} \mathtt{0}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{011}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{01111}\,R \Rightarrow_{\mathcal{G}} \mathtt{011110}$$

Se substituirmos sempre a variável *mais à direita, temos* **uma derivação pela direita**.

$$S \Rightarrow_{\mathcal{G}} \mathtt{0}\,X\,R \Rightarrow_{\mathcal{G}} \mathtt{0}\,X\,\mathtt{0} \Rightarrow_{\mathcal{G}} \mathtt{011}\,X\mathtt{0} \Rightarrow_{\mathcal{G}} \mathtt{011110}$$

Apresentámos derivações diferentes para **011110**. *Contudo, existe uma única* **árvore de derivação** *para* **011110**.



Se $\alpha \in (V \cup \Sigma)^\star$ diz-se que é uma *forma sentencial*. Uma forma sentencial α é uma `frase` (ou palavra) se consiste apenas em símbolos terminais ($\alpha \in \Sigma^\star$).

Definição 6.2 (Derivação num passo)**.** *Seja* $\mathcal{G} = (V, \Sigma, P, S)$ *e* $\alpha, \beta \in (V \cup \Sigma)^\star$.
Dada uma gramática $\mathcal{G} = (V, \Sigma, P, S)$ *uma gramática independente de contexto. Diz-se que* β *é derivável de* α *num passo e escreve-se*

$$\alpha \Rightarrow^1_{\mathcal{G}} \beta$$

se β *resulta de* α *substituindo uma ocorrência dum não terminal* A *em* α *por* γ, *onde* $A \to \gamma \in P$. *Isto é, se existem* $\alpha_1, \alpha_2 \in (V \cup \Sigma)^\star$ *tal que* $\alpha = \alpha_1 A \alpha_2$ *e* $\beta = \alpha_1 \gamma \alpha_2$.
$\Rightarrow^1_{\mathcal{G}}$ *é uma relação binária em* $(V \cup \Sigma)^\star$

Exemplo 6.14. *Sendo*
$\mathcal{G} = (\{S, A, B\}, \{a, b\}, \{S \to aB|bA, A \to a|aS|bAA, B \to b|bS|aBB\}, S)$

$$aaAb \Rightarrow^1_{\mathcal{G}} aaaSb$$

com $\alpha_1 = aa$, $\alpha_2 = b$ *e a regra* $A \to aS$.

Definição 6.3 (Derivação em n passos)**.** *Define-se a relação* $\Rightarrow^n_{\mathcal{G}}$, derivação em n passos *por indução para* $n \geq 0$*:*

- $\alpha \Rightarrow^0_{\mathcal{G}} \alpha$, *para* $\alpha \in (V \cup \Sigma)^\star$
- $\alpha \Rightarrow^1_{\mathcal{G}} \beta$, *derivação num passo*
- $\alpha \Rightarrow^{n+1}_{\mathcal{G}} \beta$, *se existe* γ *tal que* $\alpha \Rightarrow^n_{\mathcal{G}} \gamma$ *e* $\gamma \Rightarrow^1_{\mathcal{G}} \beta$

Finalmente, temos

Definição 6.4 (Derivação)**.** *A relação de* derivação $\Rightarrow^{\star}_{\mathcal{G}}$ *é o fecho reflexivo e transitivo da relação derivação num passo* $\Rightarrow^{1}_{\mathcal{G}}$*, isto é*

$$\Rightarrow^{\star}_{\mathcal{G}} = \cup_{n\geq 0} \Rightarrow^{n}_{\mathcal{G}}$$

Definição 6.5 (Linguagem gerada por uma GIC)**.** *A* linguagem gerada *por* $\mathcal{G} = (V, \Sigma, P, S)$*,* $L(\mathcal{G})$ *é o conjunto*

$$L(\mathcal{G}) = \{x \in \Sigma^{\star} | S \Rightarrow^{\star}_{\mathcal{G}} x\}$$

Uma linguagem $A \subseteq \Sigma^{\star}$ *é* independente de contexto*, se* $A = L(\mathcal{G})$ *para alguma gramática independente de contexto* $\mathcal{G}$*. Duas gramáticas* $\mathcal{G}_1$ *e* $\mathcal{G}_2$ *são* equivalentes *se* $L(\mathcal{G}_1) = L(\mathcal{G}_2)$*.*

Exemplo 6.15. *Mostrar que a linguagem* $L = \{0^n 1^n | \ n \geq 0 \ \}$ *é independebte de contexto.*
Temos que mostrar que existe uma GIC $\mathcal{G}$ *tal que* $L = L(\mathcal{G})$*.*
Seja a gramática $\mathcal{G}_2 = (\{S\}, \{0,1\}, \{S \to 0S1|\varepsilon\}, S)$
Por exemplo, $0^3 1^3 \in L(\mathcal{G}_2)$ *porque:*
$S \Rightarrow^{1}_{\mathcal{G}_2} 0S1 \Rightarrow^{1}_{\mathcal{G}_2} 00S11 \Rightarrow^{1}_{\mathcal{G}_2} 000S111 \Rightarrow^{1}_{\mathcal{G}_2} 000\epsilon 111 = 000111$ *i.e,*

$$S \Rightarrow^{4}_{\mathcal{G}_2} 000111.$$

Lema 6.1. $\forall n \geq 0, \ \ S \Rightarrow^{n+1}_{\mathcal{G}_2} 0^n 1^n$

Dem: *Por indução em* n*.* □
Vamos mostrar que $L(\mathcal{G}_2) = L$
($\subseteq$) *Pelo lema* **??** *se* $x \in L$ *então* $x \in L(\mathcal{G}_2)$*.*
($\supseteq$) *Inversamente, se* $x \in L(\mathcal{G}_2)$ *então* x *é da forma* $0^n 1^n$*, o que pode ser mostrado por indução no número de passos de derivação:*

Base. *Se* $n = 1$ *então* $x = \epsilon$

Indução. *Se* $S \Rightarrow^{n+1} x$ *então* $S \Rightarrow^{1} 0S1 \Rightarrow^{n} 0y1 = x$ *e por hipótese de indução* y *é da forma* $0^m 1^m$*. Donde* $x = 0^{m+1} 1^{m+1} \in L$

Exemplo 6.16. *Mostrar que a linguagem* $L = \{x \in \{a,b\}^{+} | \text{tem o mesmo número de a's e de b's}\}$ *é Independente de contexto.*
Seja $\mathcal{G} = (\{S, A, B\}, \{a, b\}, \{S \to aB|bA, A \to a|aS|bAA, B \to b|bS|aBB\}, S)$

Lema 6.2. $\forall x \in \{a,b\}^{+}$*,*

1. $S \Rightarrow^{\star}_{\mathcal{G}} x$ *se e só se* x *tem o mesmo número de a's e b's*
2. $A \Rightarrow^{\star}_{\mathcal{G}} x$ *se e só se* x *tem mais um a que b's*
3. $B \Rightarrow^{\star}_{\mathcal{G}} x$ *se e só se* x *tem mais um b que a's*

Dem: *Por indução em* $|x|$

Base. *Seja* $|x| = 1$. *Então* $x = a$ *ou* $= b$.

($\Leftarrow$) $A \Rightarrow_{\mathcal{G}}^{\star} a$ *e* $B \Rightarrow_{\mathcal{G}}^{\star} b$, *e não há palavras de comprimento 1 derivadas de* S.

($\Rightarrow$) *A menos de* $A \to a$ *e* $B \to b$, *todas as regras fazem aumentar o comprimento das palavras derivadas e portanto só* a *e* b *são derivadas (respectivamente de* A *e* B*)*

Indução. *Suponhamos que as afirmações são verdadeiras para* $|x| = k - 1$, *vamos ver que são verdadeiras para* $|x| = k$. **??**.

($\Rightarrow$) *Se* $S \Rightarrow_{\mathcal{G}}^{\star} x$ *então a derivação começa com a regra* $S \to aB$ *ou* $S \to bA$. *No primeiro caso,* $x = ax_1$ *e* $|x_1| = k-1$ *e* $B \Rightarrow_{\mathcal{G}}^{\star} x_1$. *Por hipótese de indução,* x_1 *tem mais um* b *que* a*'s, portanto* x *tem o mesmo número de* a*'s e de* b*'s. Analogamente se prova o segundo caso com* $x = bx_1$.

($\Leftarrow$) *Suponhamos que* x *tem o mesmo número de* a*'s e de* b*'s, queremos mostrar que* $S \Rightarrow_{\mathcal{G}}^{\star} x$. *O primeiro símbolo de* x *ou é um* a *ou é um* b. *Suponhamos que* $x = ax_1$. *Então* $|x_1| = k-1$ *e* x_1 *tem mais um* b *que* a*'s. Por hipótese de indução,* $B \Rightarrow_{\mathcal{G}}^{\star} x_1$.*Mas então,* $S \Rightarrow_{\mathcal{G}} aB \Rightarrow^{\star} ax_1 = x$. *Analogamente se prova se* $x = bx_1$. *As partes* **??** *e* **??** *provam-se de maneira análoga à parte* **??**.

□

Exercício 6.1. *Termine a demonstração anterior.* ◇

Exemplo 6.17. *Mostrar que a linguagem* $L = \{x \in \{(,)\}^{\star} | x$ *tem os parêntesis "bem casados"*$\}$ *é independente de contexto.*
Por exemplo $()()(()) \in L$ *e* $((()())) \in L$, *mas* $(()()() \notin L$, $))((\notin L$ *e* $)()(\notin L$
Informalmente $x \in L$ *se*

- *cada parêntesis esquerdo* (*emparelha com um direito*)
- *os pares de parêntesis que emparelham estão bem encaixados*

Formalmente, seja:

$$\begin{aligned} E(x) &= \text{o número de parêntesis esquerdos (em } x \\ D(x) &= \text{o número de parêntesis direitos) em } x \end{aligned}$$

Dizemos que $x \in L$ *se e só se:*

i) $E(x) = D(x)$

ii) Para qualquer prefixo y de x, $E(y) \geq D(y)$

Comecemos por ver se L é regular. Suponhamos que sim, então:

$$L \cap \{(^\star,)^\star\} = \{(^n)^n | n \geq 0\}$$

também era regular o que sabemos que é falso. O absurdo resultou de supormos que L era regular. Logo, L não é regular
Seja $\mathcal{G} = (\{S\}, \{(,)\}, \{S \to (S)|SS|\epsilon\}, S)$
Temos que $()() \in L(\mathcal{G})$: $S \Rightarrow SS \Rightarrow S(S) \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow ()(S) \Rightarrow ()()$
Vamos provar que

$$L(\mathcal{G}) = \{x \in \{(,)\}^\star | \ x \text{ satisfaz } ?? \text{ e } ??\} (= L)$$

$\subseteq$*:*
Se $S \Rightarrow_{\mathcal{G}}^\star x$, então x satisfaz **??** *e* **??**.

Lema 6.3. *$\forall \alpha \in (V \cup \Sigma)^\star$, se $S \Rightarrow_{\mathcal{G}}^\star \alpha$, então α satisfaz* **??** *e* **??**.

Pelo Lema **??**, *se $x \in L(\mathcal{G})$ então $x \in L$.*
Dem: *(do Lema* **??***) Por indução no comprimento da derivação.*

Base. *Se $S \Rightarrow_{\mathcal{G}}^0 \alpha$, então $\alpha = S$ e S verifica* **??** *e* **??**.

Indução. *Suponhamos que o resultado é válido para derivações de comprimento n. Suponhamos que $S \Rightarrow_{\mathcal{G}}^{n+1} \alpha$. Então existe $\beta \in (V \cup \Sigma)^\star$ tal que*

$$S \Rightarrow_{\mathcal{G}}^n \beta \Rightarrow_{\mathcal{G}}^1 \alpha$$

Por h.i., β satisfaz a **??** *e* **??**. *Há 3 casos a considerar, em relação à regra usada no último passo de derivação:*

$S \to \epsilon$ *Existem $\beta_1, \beta_2 \in (V \cup \Sigma)^\star$, $\beta = \beta_1 S \beta_2$ e $\alpha = \beta_1 \beta_2$. Claramente α satisfaz a* **??** *e* **??**, *uma vez que não são alterados os números nem a ordem dos parêntesis.*

$S \to SS$ *Existem $\beta_1, \beta_2 \in (V \cup \Sigma)^\star$, $\beta = \beta_1 S \beta_2$ e $\alpha = \beta_1 SS \beta_2$. Como no caso anterior, α satisfaz a* **??** *e* **??**.

$S \to (S)$ *Existem $\beta_1, \beta_2 \in (V \cup \Sigma)^\star$, $\beta = \beta_1 S \beta_2$ e $\alpha = \beta_1 (S) \beta_2$.*
$E(\alpha) = E(\beta) + 1 \geq D(\beta) + 1 = D(\alpha)$
Então, α satisfaz **??**. *Falta ver que α satisfaz* **??**.
Seja γ um prefixo de α. Queremos que $E(\gamma) \geq D(\gamma)$

- *γ é um prefixo de β_1. Então é um prefixo de β, logo verifica* **??**.

- γ *é um prefixo de* $\beta_1(S$, *mas não de* β_1.
  $E(\gamma) \;=\; E(\beta_1)+1 \;\geq\; D(\beta_1)+1 \;>\; D(\beta_1) \;=\; D(\gamma)$
- $\gamma = \beta_1(S)\delta$ *e* δ *prefixo de* β_2
  $E(\gamma) \;=\; E(\beta_1 S\delta)+1 \;\geq\; D(\beta_1 S\delta)+1 \;=\; D(\gamma)$

Em todos os casos, $E(\gamma) \geq D(\gamma)$ *e* γ *qualquer. Donde* α *satisfaz* **??**.

□

$\supseteq$:
Se x *satisfaz* **??** *e* **??**, *então* $S \Rightarrow^{\star}_{\mathcal{G}} x$.
Demonstra-se por indução em $|x|$.

Base. *Se* $|x| = 0$ *então* $x = \epsilon$ *e* $S \Rightarrow^{1}_{\mathcal{G}} \epsilon = x$.

Indução. *Suponhamos que o resultado é válido para* $|x| \leq n$. *Seja* $|x| = n+1$. *Então:*

(a) existe um prefixo próprio y *de* x *que satisfaz* **??** *e* **??**

(b) não existe tal prefixo

Caso ??:

Então $x = yz$, $0 < |z| < |x|$ *e* z *também satisfaz* **??** *e* **??**, *porque:*
$E(z) \;=\; E(x) - E(y) \;=\; D(x) - D(y) \;=\; D(z)$
E se w *é um prefixo de* z,
$E(w) \;=\; E(yw) - E(y) \;\geq\; D(yw) - D(y) \;=\; D(w)$
Por h.i., $S \Rightarrow^{\star}_{\mathcal{G}} y$ *e* $S \Rightarrow^{\star}_{\mathcal{G}} z$. *Então podemos derivar* x:

$$S \Rightarrow^{1}_{\mathcal{G}} SS \Rightarrow^{\star}_{\mathcal{G}} yS \Rightarrow^{\star}_{\mathcal{G}} yz = x$$

Caso ??:

Então, $x = (z)$ *e* z *satisfaz* **??** *e* **??**, *porque:*
$E(z) \;=\; E(x) - 1 \;=\; D(x) - 1 \;=\; D(z)$
E se w *é um prefixo não nulo de* z,
$E(w) - D(w) \;=\; E((w) - 1 - D((w) \;\geq 0$, *porque* $E((w) - D((w) \geq 1$
Então, $S \Rightarrow^{\star}_{\mathcal{G}} z$ *e*

$$S \Rightarrow^{1}_{\mathcal{G}} (S) \Rightarrow^{\star}_{\mathcal{G}} (z) = x$$

Exercício 6.2. *Construa gramáticas independentes de contexto que reconheçam as seguintes linguagens. Caso não seja indicado, o alfabeto considerado é* $\Sigma = \{0, 1\}$.

a) O conjunto vazio

b) $\Sigma^\star$

c) $\{w : w$ *contém pelo menos três 1s*$\}$

d) $\{w : w$ *começa e termina pelo mesmo símbolo*$\}$

e) $\{w : o$ *comprimento de* w *é ímpar*$\}$

f) $\{w : o$ *comprimento de* w *é ímpar e o símbolo do meio é 0*$\}$

g) $\{w : w$ *contém mais 1s que 0s*$\}$

h) $\{w : w = w^R\}$

i) $\{w : w$ *tem duas vezes mais 0s do que 1s*$\}$

j) O complemento da linguagem $\{0^n 1^n : n \geq 0\}$

k) $\{w \in \{0,1\}^\star |$ *3 divide a diferença entre o número de 0's e 1's em* $w\}$

l) $\{a^i b^j c^i \in \{a,b,c\}^\star | \, i \geq 0,\, j \geq 0\}$

m) $\{a^i b^{i+j} c^j \in \{a,b,c\}^\star | \, i \geq 0,\, j \geq 0\}$

$\diamond$

6.2.1 Derivações pela esquerda e pela direita e Árvores de derivação

Uma mesma palavra pode ter várias derivações, mas derivações em que apenas muda a ordem pela qual se fazem as substituições são iguais. Assim podemos restringir a escolha de qual a variável a substituir em cada passo:

Definição 6.6 (Derivação pela esquerda)**.** *Em cada passo de derivação substituímos o não-terminal (ou variável) mais à esquerda na forma sentencial. Designa-se por* $\overset{e}{\Rightarrow}$*.*

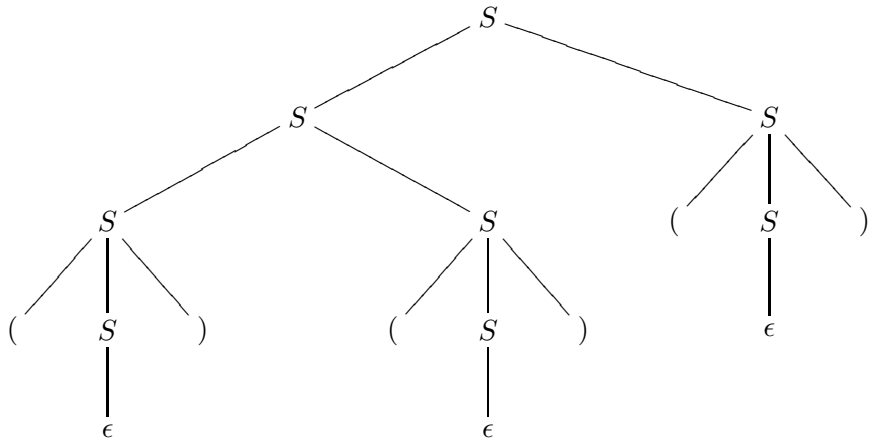
Exemplo 6.18. *No exemplo anterior, para derivar* $()()()$ *pela esquerda vem:*
$S \Rightarrow^1_{\mathcal{G}} SS \Rightarrow^1_{\mathcal{G}} SSS \Rightarrow^1_{\mathcal{G}} (S)SS \Rightarrow^1_{\mathcal{G}} ()SS \Rightarrow^1_{\mathcal{G}} ()(S)S \Rightarrow^1_{\mathcal{G}} ()()S \Rightarrow^1_{\mathcal{G}} ()()(S) \Rightarrow^1_{\mathcal{G}} ()()()$

Definição 6.7 (Derivação pela direita)**.** *Em cada passo de derivação substituímos o não-terminal (ou variável) mais à direita na forma sentencial. Designa-se por* $\overset{d}{\Rightarrow}$*.*

Exemplo 6.19. *No exemplo anterior para derivar* $()()()$ *pela direita vem:*
$S \Rightarrow^1_{\mathcal{G}} SS \Rightarrow^1_{\mathcal{G}} SSS \Rightarrow^1_{\mathcal{G}} SS(S) \Rightarrow^1_{\mathcal{G}} SS() \Rightarrow^1_{\mathcal{G}} S(S)() \Rightarrow^1_{\mathcal{G}} S()() \Rightarrow^1_{\mathcal{G}} (S)()() \Rightarrow^1_{\mathcal{G}} ()()()$

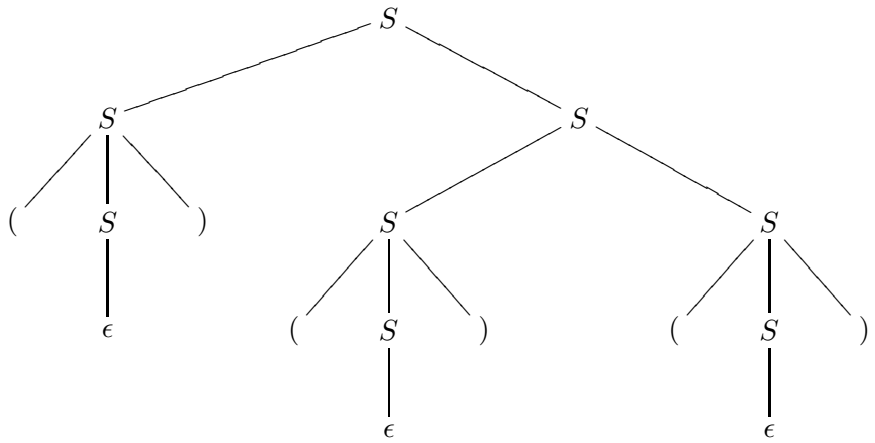
Exemplo 6.20. *Considerem-se as seguintes árvores de derivação:*

1.



$S \Rightarrow SS \Rightarrow SSS \Rightarrow (S)SS \Rightarrow ()SS \Rightarrow ()(S)S \Rightarrow ()()S \Rightarrow ()()(S) \Rightarrow ()()()$

2.

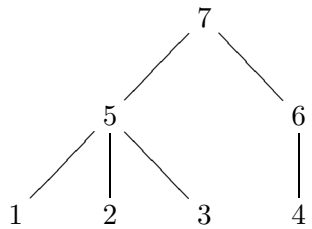


$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()SS \Rightarrow ()(S)S \Rightarrow ()()S \Rightarrow ()()(S) \Rightarrow ()()()$

Duas árvores de derivação para $()()()$ *por* $\mathcal{G}$ *que correspondem a derivações* distintas.

Recordemos que, uma árvore é grafo (não dirigido) em que o número de vértices excede em 1 o número de arcos. Um vértice é acessível a todos os outros: a *raiz*. Os vértices que são acessíveis por um arco dum outro denominam-se *filhos* desse vértice. Os vértices que não são origem de nenhum arco dizem-se *folhas*. Os restantes dizem-se *internos*.

Exemplo 6.21. *Considere a seguinte árvore ordenada:*



A raiz da árvore é 7. *Os filhos do vértice* 7 *são* 5 *e* 6, *do vértice* 5 *são* 1, 2 *e* 3 *e do vértice* 6 *é apenas* 4. *As folhas são* 1, 2, 3 *e* 4. *Os vértices* 7, 5 *e* 6 *são internos.*

Definição 6.8 (árvore de derivação)**.** *Dada uma gramática* $\mathcal{G} = (V, \Sigma, P, S)$ *uma* árvore de derivação *para* $x \in L(\mathcal{G})$ *com raiz* S *é uma árvore em que:*

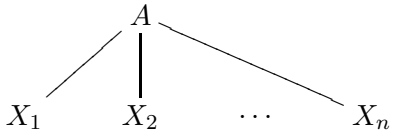
1. *A raiz é* S
2. *Cada vértice é etiquetado por um símbolo de* $V \cup \Sigma \cup \{\epsilon\}$
3. *O símbolo de um vértice interno é uma variável de* V
4. *Os símbolos das folhas são terminais ou* ϵ *(caso em que é o único filho).*
5. *Se um vértice interior tem símbolo* X *e tem* n *filhos com símbolos* $X_1, X_2, \ldots X_n$ *então existe uma produção em* P, $X \to X_1 X_2 \ldots X_n$
6. x *é a concatenação dos símbolos das folhas ordenados da esquerda para a direita*

Proposição 6.1. *Dada uma gramática* $\mathcal{G} = (V, \Sigma, P, S)$ *e* $x \in \Sigma^\star$ *são equivalentes:*

1. $S \Rightarrow_{\mathcal{G}}^\star x$, *isto é,* $x \in L(\mathcal{G})$
2. $S \overset{e}{\Rightarrow}_{\mathcal{G}}^\star x$
3. $S \overset{d}{\Rightarrow}_{\mathcal{G}}^\star x$
4. *Existe uma árvore de derivação para* x *(com raiz* S*)*

Dem: É óbvio que **??** $\Rightarrow$**??** e **??** $\Rightarrow$**??**.
Podemos generalizar a noção de árvore de derivação para o caso da raiz ser qualquer variável $A \in V$ e para que as folhas possam ser também variáveis: árvore de derivação com raiz A para $\alpha \in (V \cup \Sigma)^\star$.
?? $\Rightarrow$**??** (**??** $\Rightarrow$**??**): Vamos mostrar que para todo $A \in V$, se existe uma árvore de derivação com raiz A para uma forma sentencial α então $A \overset{e}{\Rightarrow}_{\mathcal{G}}^\star \alpha$ e $A \overset{d}{\Rightarrow}_{\mathcal{G}}^\star \alpha$.
Por indução no número de vértices internos da árvore.

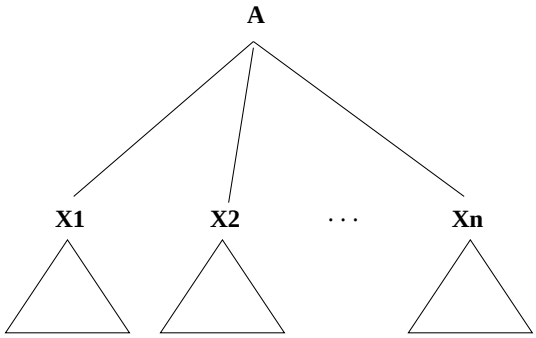
Base. Se só existe um único vértice interior a árvore é da forma:



$\alpha = X_1 X_2 \ldots X_n$ e $A \to \alpha \in P$. Então, $A \overset{d}{\Rightarrow}_{\mathcal{G}} \alpha$ e $A \overset{e}{\Rightarrow}_{\mathcal{G}} \alpha$.

Indução. • Suponhamos o resultado válido para árvores com no máximo $k-1$ vértices internos.

- Seja uma árvore de derivação com raiz A para α e com k vértices internos. Sejam $X_1, X_2, \ldots, X_n$ os filhos da raiz (e $A \to X_1 X_2 \ldots X_n \in P$).
- Cada X_i é raiz de uma árvore com menos de k vértices internos, e por h.i. $X_i \overset{e}{\Rightarrow}^{\star}_{\mathcal{G}} \alpha_i$, sendo α_i a concatenação das folhas da árvore de raiz X_i ordenadas da esquerda para a direita e $\alpha = \alpha_1 \alpha_2 \ldots \alpha_n$



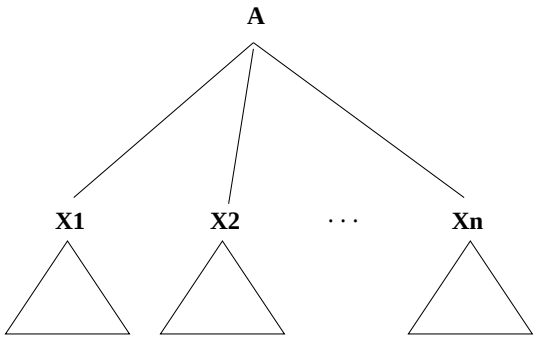
- Construímos uma derivação pela esquerda para α:
  $A \overset{e}{\Rightarrow}_{\mathcal{G}} X_1 X_2 \ldots X_n \overset{e}{\Rightarrow}^{\star}_{\mathcal{G}} \alpha_1 X_2 \ldots X_n \overset{e}{\Rightarrow}^{\star}_{\mathcal{G}} \alpha_1 \alpha_2 \ldots X_n \overset{e}{\Rightarrow}^{\star}_{\mathcal{G}} \ldots \overset{e}{\Rightarrow}^{\star}_{\mathcal{G}} \alpha_1 \alpha_2 .. \alpha_n = \alpha$
- Analogamente se constrói uma derivação pela direita, pois $X_i \overset{d}{\Rightarrow}_{\mathcal{G}} \alpha_i$

?? $\Rightarrow$**??**: por indução no número de passos da derivação:

Base. Um passo. A árvore é como no caso base anterior.

Indução.

- Suponhamos que para qualquer $A \in V$, se $A \Rightarrow^{\star} \alpha$ com menos de k passos então existe uma árvore de derivação para α com raiz A.
- Seja $A \Rightarrow^{\star} \alpha$ uma derivação com k passos. Seja o primeiro passo $A \Rightarrow X_1 X_2 \ldots X_n$.
- Qualquer símbolo de α ou é um dos X_i ou pode ser derivado de um deles. Essas derivações tem menos de k passos, logo podemos aplicar a h.i.. Isto é, existem árvores de derivação para cada uma dessas derivações.
- Construímos uma árvore de derivação para α, considerando a raiz A e os filhos as raizes dessas árvores.

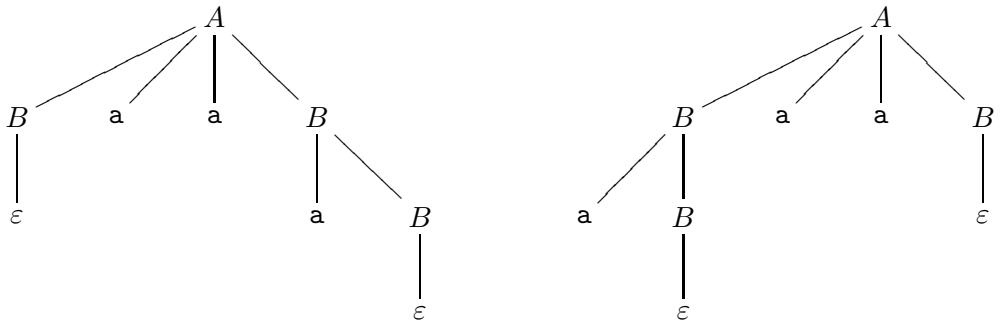


□

Exemplo 6.22. *Seja $\mathcal{G}$ a gramática* $(\{A, B\}, \{\mathtt{a}, \mathtt{b}\}, \{A \to B\mathtt{aa}B,\ B \to \mathtt{a}B,\ B \to \mathtt{b}B,\ B \to \varepsilon\}, A)$.
Duas derivações pela esquerda para aaa*:*

$$A \Rightarrow B\mathtt{aa}B \Rightarrow \varepsilon\mathtt{aa}B \Rightarrow \mathtt{aaa}B \Rightarrow \mathtt{aaa}\varepsilon = \mathtt{aaa}$$
$$A \Rightarrow B\mathtt{aa}B \Rightarrow \mathtt{a}B\mathtt{aa}B \Rightarrow \mathtt{a}\varepsilon\mathtt{aa}B \Rightarrow \mathtt{aaa}\varepsilon = \mathtt{aaa}$$

Duas árvores de derivação para **aaa***:*

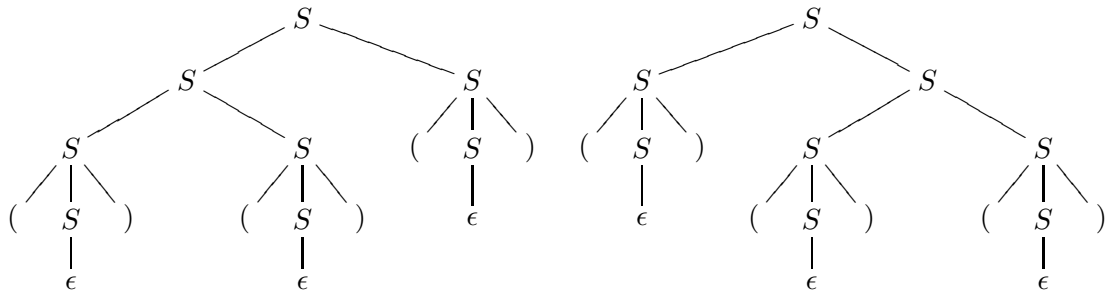


6.3 Ambiguidade em Gramáticas e em Linguagens

Definição 6.9. *Uma gramática $\mathcal{G}$ diz-se* ambígua *se existe mais do que uma árvore de derivação para algum $x \in L(\mathcal{G})$. Eequivalemente se existe mais do que uma derivação pela esquerda (ou pela direita) para algum $x \in L(\mathcal{G})$.)*

Exemplo 6.23. *A gramática $\mathcal{G} = (\{S\}, \{(,)\}, \{S \to (S)|SS|\epsilon\}, S)$ dada para a linguagem L de parentesis "bem casados" é ambígua. $()()() \in L(\mathcal{G})$ tem duas árvores de derivação*

distintas:



Agora, se substituirmos as regras de $\mathcal{G}$ por:

$$\begin{aligned} S &\rightarrow XS|\epsilon \\ X &\rightarrow (S) \end{aligned}$$

obtemos uma gramática que gera L e não é ambígua.

Exercício 6.3. *Verifique que ()()() só tem uma árvore de derivação.* $\diamond$

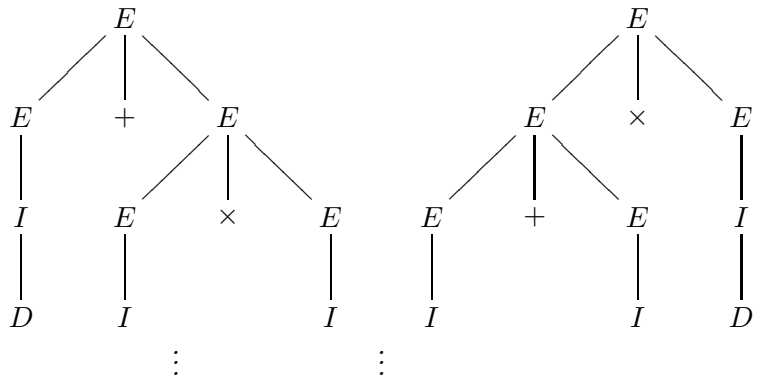
Para mostrar que uma gramática não é ambígua, temos mostrar que *nenhuma* palavra gerada tem duas árvores de derivação (ou duas derivações pela esquerda distintas)

Exemplo 6.24. *Consideremos a gramática obtida no exemplo anterior.*
As derivações a partir de S são (ou ϵ) ou da forma:
$S \Rightarrow^1 XS \Rightarrow^1 XXS \Rightarrow^1 \ldots \Rightarrow^1 X^i$, $i > 0$
e a partir de X temos univocamente, para algum $j \geq 0$, $X \Rightarrow^{j+1} (X^j)$.
Cada palavra gerada é univocamente determinada pela escolha de um i e, para cada i, um j_i . O que prova que não é ambígua.

Exemplo 6.25. *A gramática já descrita para a linguagem das expressões aritméticas é ambígua:*

$$\begin{aligned} \mathcal{G}_e \;=\; (\{E,D,I\}, \{0,1,2,3,4,5,6,7,8,9,\times,+,(,)\}, & \\ E &\rightarrow E\times E|E+E|(E)|I \\ \{\; I &\rightarrow D|DI \qquad\qquad\quad \} \\ D &\rightarrow 0|1|2|3|4|5|6|7|8|9 \\ & \qquad\qquad\qquad\qquad\quad , E) \end{aligned}$$

E estas regras para E não capturam a precedência da multiplicação em relação à adição: por exemplo, existem duas árvores de derivação para $2+5\times 4$: uma correspondendo ao valor de $2+(5\times 4)$ e outra ao de $(2+5)\times 4$:



Isto é mau se pensarmos implementar um avaliador de expressões aritméticas... A ambiguidade vem de:

- *a precedência dos operadores não é respeitada*
- *Uma sequência de operações iguais pode ser agrupada à esquerda ou à direita.*

Exercício 6.4. *Determinar duas árvores de derivação para* $2 + 4 + 5$ $\diamond$

Exemplo 6.26. *Vamos ver como podemos remover a ambiguidade da gramática do exemplo anterior. Podemos solucionar o problema introduzindo mais não terminais, que correspodam ao agrupamento de operações pretendido:* $\times$ *maior precedência que* $+$ *e associatividade à esquerda para operadores iguais. Vamos considerar mais dois não-terminaias:*

`F` **factores** *identificadores ou expressões entre parentisis:* $F \to I|(E)$

`T` **termos** *produto de um ou mais factores; não pode ser decomposto por operadores* `+`*:* $T \to F|T \times F$

Assim, se se substituir as regras para E pelas seguintes, a gramática resultante não é ambígua:

$$\begin{aligned} E &\to E + T|T \\ T &\to T \times F|F \\ F &\to (E)|I \end{aligned}$$

Exercício 6.5. *Verificar que agora* $2 + 5 \times 4$ *só admite uma árvore de derivação.* $\diamond$

Em geral, não é fácil mostrar que uma gramática não é ambígua (temos que mostrar que todas as palavras da linguagem tem uma única árvore de derivação).

Exemplo 6.27. *Neste caso deve-se a:*

- *facilmente se vê que um factor só tem uma árvore de derivação.*

- *as palavras derivadas de T são produtos de factores tal que se $f_1 \times f_2 \times \ldots f_{n-1} \times f_n$ é derivada então a única árvore de derivação é tal que os filhos da raiz são T, $\times$, F onde T é a raiz duma árvore de derivação para $f_1 \times f_2 \times \ldots f_{n-1}$ e F raiz duma árvore de derivação para f_n.*

- *do mesmo modo se prova que só existe uma árvore de derivaçã*

  para as palavras derivadas a partir de E: expressões que são sequências de termos ligados por o operador $+$

Note que se uma gramática não for ambígua cada palavra tem apenas uma derivação pela esquerda (e só uma derivação para a direita)
Embora possa ser díficil mostrar que uma gramática não é ambígua, estas gramáticas são muito importantes:

- em compiladores e outros avaliadores de expressões, as gramáticas permitem construir estruturas sintácticas (correspondentes a árvores de derivação) a partir das quais é determinado o significado das expressões: se a gramática é ambígua então uma mesma expressão (p.e programa) pode ser avaliada várias maneiras (no caso do programa, produzir mesmo resultados diferentes).

Exemplo 6.28. *A gramática $\mathcal{G}$ dada por*

$$(\{A, B\}, \{\mathtt{a}, \mathtt{b}\}, \{A \to B\mathtt{aa}B,\ B \to \mathtt{a}B,\ B \to \mathtt{b}B,\ B \to \varepsilon\}, A)$$

e a gramática $\mathcal{G}' = (\{A, B, C\}, \{\mathtt{a}, \mathtt{b}\}, P, A)$ em que P é

$$A \ \to \ C\,\mathtt{aa}\,B$$

$$\begin{array}{lcllcl} B & \to & \mathtt{a}\,B & C & \to & \mathtt{b}\,C \\ B & \to & \mathtt{b}\,B & C & \to & \mathtt{ab}\,C \\ B & \to & \varepsilon & C & \to & \varepsilon \end{array}$$

geram a linguagem das palavras em $\{\mathtt{a}, \mathtt{b}\}^\star$ que têm $\mathtt{aa}$ como subpalavra. Mas $\mathcal{G}$ é ambígua, enquanto que $\mathcal{G}'$ 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

$$x_1\mathtt{aa}x_2 \quad com \quad x_1 \in \{\mathtt{b}, \mathtt{ab}\}^\star,\ \ x_2 \in \{\mathtt{a}, \mathtt{b}\}^\star$$

enquanto que $\mathcal{G}$ decompõe em

$$x_1\mathtt{aa}x_2 \quad com \quad x_1, x_2 \in \{\mathtt{a}, \mathtt{b}\}^\star$$

Exercício 6.6. *Considere a gramática*
$G = (\{S, X\}, \{S \to X \mid Sa, X \to a \mid b\}, \{a, b\}, S)$
Mostre que não é ambígua e determina a linguagem gerada. $\diamond$

Resolução ??
Pela primeira produção de S apenas se deriva a e b e univocamente.
A segunda produção de S pode ser aplicada um certo número de vezes $i \geq 0$ e depois aplicar a primeira produção: $S \Rightarrow Sa \Rightarrow Saa \Rightarrow \ldots \Rightarrow Sa^i \Rightarrow Xa^i$
Isto é, $S \Rightarrow^{i+1} Xa^i$.
Para cada forma sentencial Xa^i só há duas produções alternativas a aplicar, produzindo a^{i+1} ou ba^i. Temos que a linguagem gerada é:
$L = \{a^i \mid i \geq 1\} \cup \{ba^i \mid i \geq 0\}$
Uma palavra gerada é univocamente determinada pela escolha de um i e de uma das duas produções no último passo.

Exercício 6.7. *Considere a gramática*
$G = (\{S\}, \{S \to aSbS \mid bSaS \mid \epsilon\}, \{a, b\}, S)$

a) Mostre que $L(G)$ é o conjunto de palavras de $\{a, b\}^\star$ com o mesmo número de a's e b's.

b) Mostre que G é ambígua.

c) Escreva uma gramática não ambígua equivalente a G

$\diamond$

Definição 6.10. *Uma linguagem independente de contexto diz-se (inerentemente)* ambígua *se todas as gramáticas independentes de contexto que a geram são ambíguas.*

Exemplo 6.29. *A linguagem independente de contexto*

$$L = \{a^n b^n c^m d^m | n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n | n \geq 1, m \geq 1\}$$

é ambígua.
Descreva uma gramática para L e verifica que é ambígua (porque todas são!).

Normalmente não é fácil provar que uma linguagem é ambígua, até, no caso geral, é um problema indecidível. Mas, uma linguagem deste tipo não seria boa para descrever a sintaxe duma linguagem de programação.

6.4 Algumas Propriedades das L. I. C.

Começamos por mostrar que a classe das linguagens independentes de contexto é fechada para a união finita, a concatenação e o fecho de Kleene (mais adiante, vamos ver que não é fechada para a complementação nem para a intersecção).
Em seguida provamos que a classe das linguagens regulares está contida propriamente (ou seja, está contida e é diferente) na classe das linguagens independentes de contexto. Tal prova segue quase como corolário do facto da classe das linguagens independentes de contexto ser fechada para as operações atrás referidas. Apresentamos ainda uma prova alternativa, na qual se descreve um método para determinar uma gramática independente de contexto que gera a linguagem aceite por um dado autómato finito.

6.4.1 Propriedades de Fecho

Vamos ver classe das linguagens independentes de contexto é fechada para a reunião finita, concatenação finita e fecho de Kleene.

Proposição 6.2. *A reunião finita de linguagens independentes de contexto é uma linguagem independente de contexto.*

Dem: Sejam L_1 e L_2 linguagens independentes de contexto (**LICs**) de alfabeto Σ.

Base. Sejam G_1 e G_2 gramáticas independentes de contexto que geram L_1 e L_2, respectivamente.

$$G_1 = (V_1, \Sigma, P_1, S_1) \text{ e } G_2 = (V_2, \Sigma, P_2, S_2)$$

Sem perda de generalidade, vamos supor que $V_1 \cap V_2 = \emptyset$ (se $V_1 \cap V_2 \neq \emptyset$ escolhemos outros símbolos para representar as variáveis comuns).

Consideremos a gramática $G = (V, \Sigma, P, S)$ em que: Seja $G = (V, \Sigma, P, S)$ em que

- S uma variável nova
- $V = \{S\} \cup V_1 \cup V_2$
- $P = \{S \to S_1,\ S \to S_2\} \cup P_1 \cup P_2$

Não é dificil concluir que $L_1 \cup L_2 = \mathcal{L}(G)$. De facto, $x \in \mathcal{L}(G)$ 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 $S_2 \Rightarrow^\star x$.

Indução. Supomos agora, *como hipótese de indução* 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. Para concluir que L é uma LIC, basta notar que tal união é união de uma LIC com n LICs. Como

por hipótese a união de n LICs é uma LIC, conclui-se que L é união de duas LICs, o que (vimos acima) é uma LIC.

□

Proposição 6.3. *A concatenação de duas linguagens independentes de contexto é uma linguagem independente de contexto.*

Dem: 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, respectivamente.
Sem perda de generalidade, supomos $V_1 \cap V_2 = \emptyset$. Seja S uma nova variável. A linguagem L_1L_2 é a linguagem gerada pela gramática independente de contexto $G = (V, \Sigma, P, S)$ em que:

- S uma nova variável
- $V = \{S\} \cup V_1 \cup V_2$
- $P = \{S \to S_1S_2\} \cup P_1 \cup P_2$

Vamos ver que $L_1L_2 = \mathcal{L}(G)$:
$x \in \mathcal{L}(G)$ sse $S \Rightarrow^\star x$. Mas, $S \Rightarrow^\star x$ se e só se $S \Rightarrow S_1S_2$, $S_1 \Rightarrow^\star x_1$, $S_2 \Rightarrow^\star x_2$ e $x = x_1x_2$. □

Exercício 6.7.1. Mostre que a concatenação finita de linguagens independentes de contexto é independente de contexto.

Proposição 6.4. *O fecho de Kleene duma linguagem independente de contexto é independente de contexto.*

Dem: Seja L uma LIC, e seja $G_1 = (V_1, \Sigma, P_1, S_1)$ uma GIC (gramática independente de contexto) que gera L. Seja $G = (V, \Sigma, P, S)$ em que

- S uma nova variável
- $V = \{S\} \cup V_1$
- $P = \{S \to S_1S, S \to \epsilon\} \cup P_1$

Vamos ver que $L_1{}^\star = \mathcal{L}(G)$:

$\mathcal{L}(G) \subseteq L^\star$**.** Com efeito, qualquer que seja $n \geq 1$ tem-se $S \Rightarrow^n x$ *sem aplicar regras em* P_1 se e só se $x = S_1^nS$ 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 $x = \varepsilon$. Em conclusão, $\mathcal{L}(G) \subseteq L^\star$.

$\mathcal{L}(G) \supseteq L^\star$**.** Reciprocamente, $\mathcal{L}(G) \supseteq L^\star$ pois 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 $S \Rightarrow^{k+1} S_1^k \Rightarrow^\star x_1S_1^{k-1} \Rightarrow^\star x$.

□

6.4.2 Linguagem regulares e gramáticas lineares

Proposição 6.5. *Qualquer linguagem regular é independente de contexto.*

Dem: Vamos mostrar a proposição por indução sobre o número de operadores numa expressão regular que descreva a linguagem dada.
Seja $L \subseteq \Sigma^\star$ uma linguagem regular, e seja r uma expressão regular que descreve L.
Caso r tenha 0 operadores, então

- se $r = \varepsilon$, seja $G = (\{S\}, \Sigma, \{S \to \varepsilon\}, S)$;
- se $r = \emptyset$, seja $G = (\{S\}, \Sigma, \{\ \}, S)$;
- se $r = a$, $a \in \Sigma$ seja $G = (\{S\}, \Sigma, \{S \to a\}, S)$.

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. *Se r tem algum operador*, então $r = (r_1^\star)$, ou $r = (r_1 r_2)$ $r = (r_1 + r_2)$. Tendo r_1 e r_2 menos operadores do que r, por hipótese, $\mathcal{L}(r_1)$ e $\mathcal{L}(r_2)$ são LICs.
Então, como já 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 $\mathcal{L}(r) = \mathcal{L}(r_1) \cup \mathcal{L}(r_1)$, respectivamente. □
Podemos também mostrar a Proposição **??**, definindo uma GIC naturalmente associada a um autómato finito determinístico que reconhece a linguagem dada.
Dem: (**outra demonstração da Proposição ??**) Seja L uma linguagem regular, e seja $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ um autómato finito determinístico tal que $L = \mathcal{L}(\mathcal{A})$.
A gramática $G = (\mathcal{V}, \Sigma, P, V_{s_0})$ vai ser assim definida:

- A cada *estado* $s \in S$ associa-se uma *variável* V_s.

$$\mathcal{V} = \{V_s \mid s \in S\}$$

- O conjunto de regras P é assim definido:
  - A cada *transição* $\delta(s, a) = s'$ faz-se corresponder uma *regra* $V_s \to aV_{s'}$;
  - Para todo $s_f \in F$ inclui-se ainda a regra $V_{s_f} \to \varepsilon$.

Vamos ver que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(G)$.
I.e , $\forall x \in \Sigma^\star$, $\widehat{\delta}(s_0, x) \in F$ sse $V_{s_0} \Rightarrow_G^\star x$
Comecemos por ver que:
$\forall s \in S \forall w \in \Sigma^\star$ $\widehat{\delta}(s, w) = s'$ sse $V_s \Rightarrow^\star wV_s'$

Base. $w = \epsilon$, então $\widehat{\delta}(s, \epsilon) = s$ e $V_s \Rightarrow^0 V_s$

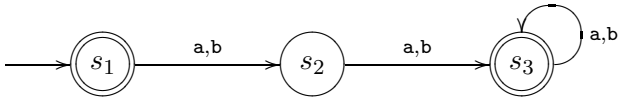
Indução. $w = xa$ e por h.i. $\widehat{\delta}(s, x) = s'$ sse $V_s \Rightarrow^\star xV'_s$.

$\widehat{\delta}(s, w) = \widehat{\delta}(s, xa) = \delta(\widehat{\delta}(s, x), a) = \delta(s', a) = s''$, para algum $s'' \in S$, sse $V_s \Rightarrow^\star xV'_s \Rightarrow xaV''_s = wV''_s$ porque $V'_s \to aV''_s \in P$

Temos então que:

$$\begin{aligned} x \in L(A) &\iff \widehat{\delta}(s_0, x) \in F \\ &\iff \widehat{\delta}(s_0, x) = s_f, \;\; s_f \in F \\ &\iff V_{s_0} \Rightarrow^\star xV_{s_f} \Rightarrow x \\ &\iff x \in L(G) \end{aligned}$$

Exemplo 6.30. *Considere o autómato seguinte*



A gramática definida na prova da proposição anterior 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:*

$$\begin{aligned} V_{s_1} &\to \varepsilon \mid \mathtt{a}V_{s_2} \mid \mathtt{b}V_{s_2} \\ V_{s_2} &\to \mathtt{a}V_{s_3} \mid \mathtt{b}V_{s_3} \\ V_{s_3} &\to \varepsilon \mid \mathtt{a}V_{s_3} \mid \mathtt{b}V_{s_3} \end{aligned}$$

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 6.11. *Uma gramática independente de contexto diz-se* linear à esquerda *se todas as suas regras são da forma* $X \to w$ *ou* $X \to Yw$*, em que* X *e* Y *são variáveis e* $w \in \Sigma^\star$*.*
Uma gramática independente de contexto diz-se 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* $w \in \Sigma^\star$*.*

Às gramáticas lineares à direita ou lineares à esquerda também se chama **gramáticas regulares**. Essa designação surge naturalmente como consequência dos resultados seguintes.

Proposição 6.6. *A classe de linguagens regulares é a classe de linguagens independentes de contexto geradas por gramáticas lineares à direita.*

Dem: Da prova da Proposição **??** (segunda versão) conclui-se que a classe de linguagens regulares está contida na classe de linguagens geradas por gramáticas lineares à direita.
Resta mostrar que "a classe de linguagens regulares contém a classe de LICs geradas por GICs lineares à direita".
$\subseteq$: Seja $L = L(G)$ para uma gramática linear à direita $G = (V, \Sigma, P, S)$. Construímos um AFND-ϵ, $A = (Q, \Sigma, \delta, s_S, \{s_\epsilon\})$ que simula as derivações em G.

- $Q = \{s_\alpha \,|\alpha$ é S ou um sufixo de qualquer lado direito duma produção de $P\}$
- Define-se δ por:
  1. Se $X \in V$, então $\delta(s_X, \epsilon) = \{s_\alpha \,|X \to \alpha \in P\}$
  2. Se $a \in \Sigma$ e $\alpha \in \Sigma^\star \cup \Sigma^\star V$, então $\delta(s_{a\alpha}, a) = \{s_\alpha\}$

Mostra-se por indução no número de passos da derivação ou transições que
$s_\alpha \in \widehat{\delta}(s_S, w)$ se e só se $S \Rightarrow^\star xX \Rightarrow xy\alpha = w\alpha$ onde $X \to y\alpha$ e $w = xy$; ou se $\alpha = S$ e $w = \epsilon$.
Como s_ϵ é o único estado final, A aceita w se e só se $S \Rightarrow^\star xX \Rightarrow w$. O que equivale a dizer que $w \in L(G)$.

Exercício 6.8. *Termine a demonstração anterior.* $\diamond$

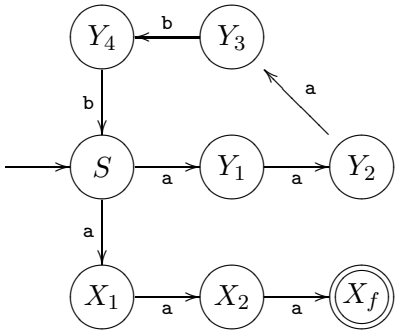
Exemplo 6.31. *Seja* $\mathcal{G} = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, \{S \to \mathtt{aaa},\ S \to \mathtt{aaabb}S\}, S)$.
A gramática $\mathcal{G}'$ *é equivalente a* $\mathcal{G}$ *e está nas condições impostas na prova anterior:*

$$\mathcal{G}' = (\{S, X_1, X_2, X_3, Y_1, Y_2, Y_3, Y_4, X_f\}, \{\mathtt{a}, \mathtt{b}\}, P, S)$$

em que as regras são

$$\begin{array}{lcllcl} S & \to & \mathtt{a}X_1 & S & \to & \mathtt{a}Y_1 \\ X_1 & \to & \mathtt{a}X_2 & Y_1 & \to & \mathtt{a}Y_2 \\ X_2 & \to & \mathtt{a}X_f & Y_2 & \to & \mathtt{a}Y_3 \\ & & & Y_3 & \to & \mathtt{b}Y_4 \\ & & & Y_4 & \to & \mathtt{b}S \\ X_f & \to & \varepsilon & & & \end{array}$$

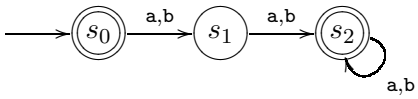
O autómato finito que se obtém é:



Exercício 6.9.

1. *Construa uma gramática linear à direita que gere a linguagem aceite pelo autómato* $A = (\{s_0, s_1, s_2\}, \{a, b\}, \delta, \{s_0, s_2\})$ *onde*

	a	b
s_0	s_1	s_1
s_1	s_2	s_2
s_2	s_2	s_2



2. *Construa um AFND's correspondente à gramática cujas regras são:* $\{S \to 0A,\ A \to 10A \mid \epsilon\}$

Resolução ??
$A = (\{s_S, s_{0A}, s_A, s_\epsilon, s_{10A}\}, \{0, 1\}, \delta, s_S, \{s_\epsilon\})$, *onde*

$$\begin{aligned} \delta(s_S, \epsilon) &= \{s_{0A}\} \\ \delta(s_A, \epsilon) &= \{s_{10A}, s_\epsilon\} \\ \delta(s_{0A}, 0) &= \{s_A\} \\ \delta(s_{10A}, 1) &= \{s_{0A}\} \end{aligned}$$

◇

Seja x uma qualquer palavra de alfabeto Σ. 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 $L^r = \{x^r \mid x \in L\}$.

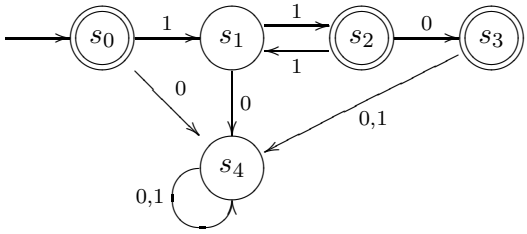
Lema 6.4. *Qualquer que seja a linguagem L, tem-se L é regular se e só se L^r é regular.*

Dem: Se mostrarmos que *se L for regular então L^r é regular*, podemos concluir que se L^r for regular então $(L^r)^r$ é regular. Mas como, $(L^r)^r = L$, fica provado que se L^r for 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 $s \in S$

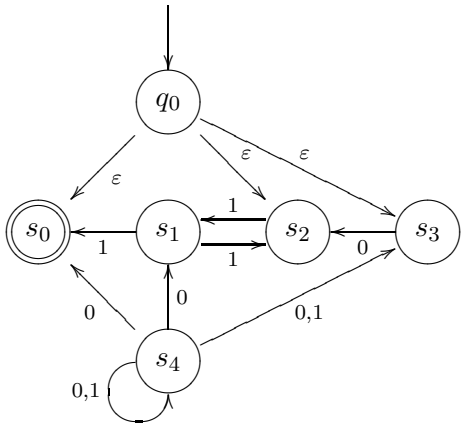
e símbolos $a \in \Sigma$, tem-se $\delta'(s', a) = \{s\}$ sse $\delta(s, a) = s'$ (ou seja, orientam-se os arcos em sentido inverso).

Não é difícil concluir que existe caminho em $\mathcal{A}$ do estado s_0 para algum dos estados finais etiquetado por x se e só se existe caminho em $\mathcal{A}'$ do estado q_0 para o estado s_0 etiquetado por x^r. Em $\mathcal{A}'$, efectua-se transição por ε para o estado final referido, e efectuam-se as transições (inversas) em $\mathcal{A}'$ pela ordem inversa das efectuadas no processamento de x por $\mathcal{A}$. □

Exemplo 6.32. *Seja L a linguagem reconhecida pelo autómato*



A linguagem L^r é a linguagem reconhecida pelo autómato



Proposição 6.7. *A classe de LICs geradas por GICs lineares à direita é a classe de LICs geradas por GICs lineares à esquerda.*

Prova. Já vimos que a classe das linguagens regulares é a classe das linguagens geradas por gramáticas lineares à direita.

Por outro lado, 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 $X \to w^r Y$ 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. Assim, podemos concluir, pelo lema anterior, que L^r é regular. Mas se L^r é regular também L é regular. Em conclusão, *qualquer linguagem que seja gerada por uma gramática linear à esquerda, pode ser gerada por uma gramática linear à direita.*

De modo análogo se mostra que *se L é gerada por uma GIC linear à direita então L^r é gerada por uma GIC linear à esquerda.*

Assim, como $(L^r)^r = L$, concluimos ainda que L é gerada por uma GIC linear à esquerda se e só se L^r é gerada por uma gramática linear à direita.

Finalmente, pelo Lema e Proposição anteriores, L^r é gerada por uma GIC linear à direita se e só L é gerada por uma GIC linear à direita. Logo, L é gerada por uma GIC linear à esquerda se e só se L é gerada por uma gramática linear à direita. □

Já vimos que existem linguagens independentes de contexto que não são regulares. Por exemplo, $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$ de alfabeto $\{\mathtt{0},\mathtt{1}\}$, as sequências de parentesis curvos que são bem formadas, etc. Com o exemplo seguinte pretendemos ilustrar como é que se pode tentar provar que uma dada gramática gera uma certa linguagem.

Exemplo 6.33. *A linguagem* $L = \{\mathtt{aaa}\,x\,\mathtt{a}\,y \mid |x| \leq 2|y|, \quad x,y \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star\}$ *de alfabeto* $\{\mathtt{0},\mathtt{1},\mathtt{b},\mathtt{a}\}$ *não é regular mas é independente de contexto.*

L **não é regular**, *pois não satisfaz a condição do Lema da repetição (para linguagens regulares).*
De facto, seja $n \in \mathbb{N} \setminus \{0\}$ qualquer. *Procuremos uma palavra* $x \in L$ *tal que* $|x| \geq n$ *e quaisquer que sejam* u,v,w *com* $x = uvw$, $|uv| \leq n$, $v \neq \varepsilon$, *existe* $i \in \mathbb{N}$ *tal que* $uv^iw \notin L$.
Uma ideia... *por exemplo,* $x = \mathtt{aaa1}^{2n}\mathtt{a1}^n$ *(não pode ter mais* `1`*'s na primeira parte)*
Para que $|uv| \leq n$ *e* $v \neq \varepsilon$ *é necessário que* uv *seja prefixo de* $\mathbf{aaa1^{n\text{-}3}}$ *se* $n \geq 3$ *ou de* **aaa** *se* $n < 3$. *Então* v *tem algum* `a` *ou* v *não tem* `a`*'s.*
Se v tiver algum `a`, *corta-se* v *(toma-se* $i = 0$*) e* $uv^0w \notin L$ *porque não começa por* `aaa`.
Se v não tiver `a`*'s então só tem* `1`*'s, sendo*

$$x = \underbrace{\mathtt{aaa1}^k}_{u}\ \underbrace{\mathtt{1}^{|v|}}_{v}\ \underbrace{\mathtt{1}^{2n-k-|v|}\mathtt{a1}^n}_{w}$$

Nesse caso, por exemplo $uvvw \notin L$ *(caso* $i = 2$*)*

$L = \{\mathtt{aaa}\,x\,\mathtt{a}\,y \mid |x| \leq 2|y|, \quad x,y \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star\}$ **é independente de contexto** *porque é gerada, por exemplo, pela gramática* $\mathcal{G} = (\{S,R,A\},\{\mathtt{0},\mathtt{1},\mathtt{a},\mathtt{b}\},P,S)$ *cujas regras são*

$$\begin{array}{lcl} S & \to & \mathtt{aaa}\,R \\ R & \to & A\,A\,R\,A \\ R & \to & A\,R\,A \\ R & \to & R\,A \\ R & \to & \mathtt{a} \\ A & \to & \mathtt{b} \mid \mathtt{1} \mid \mathtt{0} \end{array}$$

Vamos mostrar formalmente que $\mathcal{G}$ gera L, ou seja $\mathcal{L}(\mathcal{G}) = L$.
Mostremos, por indução sobre o número de regras aplicadas, que: quaisquer que sejam $n \in \mathbb{N} \setminus \{0\}$ e $x \in (V \cup \Sigma)^\star$, se $R \Rightarrow^n x$ sem substituir A 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 $0 \leq k \leq 2n$. (Observe que, como consequência, qualquer palavra em $\Sigma^\star$ gerada a partir de R é da forma $x\mathtt{a}y$ com $|x| \leq 2|y|$, $x, y \in \{\mathtt{0}, \mathtt{1}, \mathtt{b}\}^\star$.)*

Base. *Caso $n = 1$. Se $R \Rightarrow^1 x$ então $x = A^0 \,\mathtt{a}\, A^0$, ou $x = A^k \, R \, A$ com $0 \leq k \leq 2$ pois*

$$R \Rightarrow A^k \, R \, A \quad 0 \leq k \leq 2 \qquad \textit{por} \begin{cases} R \to A \, A \, R \, A \\ R \to A \, R \, A \\ R \to R \, A \end{cases}$$

ou

$$R \Rightarrow A^k \,\mathtt{a}\, A^0 \quad 0 \leq k \leq 0 \qquad \textit{por} \begin{cases} R \to \mathtt{a} \end{cases}$$

Indução. *Se $n \geq 1$, então por definição de $\Rightarrow^{n+1}$ tem-se*

$$R \Rightarrow^{n+1} x \quad \textit{sse} \quad \exists x' \in (V \cup \Sigma)^\star \;\; (R \Rightarrow^n x' \;\wedge\; x' \Rightarrow x)$$

Para $R \Rightarrow^{n+1} x$ sem substituir A's, então na derivação de x' não se substituiu qualquer A.

Supondo, como hipótese de indução, que qualquer que seja $w \in (V \cup \Sigma)^\star$, se $R \Rightarrow^n w$ sem substituir *A, então*

$$w = A^k R A^n, \;\; \textit{com } 0 \leq k \leq 2n$$

ou

$$w = A^k \mathtt{a} A^{n-1}, \;\; \textit{com } 0 \leq k \leq 2(n-1),$$

*podemos deduzir (*porquê?*) que*

$$x' = A^k R A^n, \quad \textit{para algum } k \textit{ tal que } 0 \leq k \leq 2n$$

Como x resulta de $x' = A^k R A^n$ por aplicação duma regra, x **é da forma esperada**

$$x = A^q \,\mathtt{a}\, A^n \qquad e \quad 0 \leq q \leq 2n$$

ou

$$x = A^q \, R \, A^{n+1} \quad e \quad 0 \leq q \leq 2n+2$$

De facto,

$x = A^{k+2}RA^{n+1}$ *por* $A^kRA^n \Rightarrow A^kAARA^{n+1}$, *ou*

$x = A^{k+1}RA^{n+1}$ *por* $A^kRA^n \Rightarrow A^kARA^{n+1}$, *ou*

$x = A^kRA^{n+1}$ *por* $A^kRA^n \Rightarrow A^kRA^{n+1}$, *ou*

$x = A^k\mathtt{a}A^n$ *por* $A^kRA^n \Rightarrow A^k\mathtt{a}A^n$,

Como $0 \leq k \leq 2n$ *então*

$$\begin{aligned} x &= A^q\,R\,A^{n+1} \quad e \quad 0 \leq q \leq k+2 \leq 2(n+1) \\ &\text{ou} \\ x &= A^q\,\mathtt{a}\,A^n \qquad e \quad 0 \leq q \leq 2n \end{aligned}$$

Concluimos que: qualquer palavra em $\Sigma^\star$ *gerada a partir de* R *é da forma* $x\mathtt{a}y$ *com* $|x| \leq 2|y|$, $x, y \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star$.
Resta mostrar que toda a palavra da forma $x\mathtt{a}y$ *com* $|x| \leq 2|y|$, *com* $x, y \in \{\mathtt{0},\mathtt{1},\mathtt{b}\}^\star$ *pode ser derivada a partir de* R. *De facto,*

- *Se* $|x| > |y|$, *aplicamos* $|x|-|y|$ *vezes* $R \to A\,A\,R\,A$ *seguido de* $2|y|-|x|$ *vezes* $R \to A\,R\,A$ *para obter* $A^{|x|}\,R\,A^{|y|}$, *depois* $R \to \mathtt{a}$, *e finalmente* $|x|+|y|$ *vezes regras para* A *para derivar* x *e* y. *Ou seja, se* $k = |x|$ *e* $n = |y|$

$$R \Rightarrow^{k-n} \quad A^{2(k-n)}\,R\,A^{k-n} \quad \Rightarrow^{2n-k} \quad A^k\,R\,A^n$$

- *Se* $|x| < |y|$, *aplicamos* $|x|$ *vezes* $R \to A\,R\,A$ *e* $|y|-|x|$ *vezes* $R \to R\,A$, *e depois* $R \to \mathtt{a}$, *e finalmente* $|x|+|y|$ *vezes regras para* A *para derivar* x *e* y.

$$R \Rightarrow^{k} \quad A^k\,R\,A^k \quad \Rightarrow^{n-k} \quad A^k\,R\,A^n$$

- *E se* $|x| = |y|$*? (*Queira o leitor interessado preencher os detalhes... *)*

Conclui-se que $\mathcal{L}(\mathcal{G}) = L$, *isto é que*

$$\begin{aligned} &(i) \quad \forall x \in L \quad S \Rightarrow^\star x \\ &(ii) \quad \forall x \in \Sigma^\star \quad \text{se } S \Rightarrow^\star x \text{ então } x \in L \end{aligned}$$

6.5 Formas normais

Em várias situações é útil que as regras das gramáticas tenham formas especiais Para tal é necessário que se possa transformar qualquer gramática numa equivalente e que esteja em

forma normal. As formas normais que vamos considerar não geram a palavra vazia (notar que isto não é uma restrição demasiado forte...). Para as linguagens que não gerem ϵ temos, por exemplo,:

Forma normal de Chomsky (FNC) as produções são da forma $A \to BC$ ou $A \to a$, para $A, B, C \in V$ e $a \in \Sigma$

Uma vantagem é que as árvores de derivação são todas binárias.

Forma normal de Greibach (FNG) as produções são da forma

$$A \to aB_1B_2 \dots B_k$$

para algum $k \geq 0$ e $A, B_1, \dots B_k \in V$ e $a \in \Sigma$.

Uma vantagem é que em cada passo de derivação um terminal é consumido. Isto é, se $|x| = n$ e $x \in L(G)$ então $S \Rightarrow^n x$.

Exemplo 6.34. *Consideremos mais uma vez a linguagem, dos parêntesis bem casados. Já vimos que uma gramática para esta linguagem pode ser:*

$$G = (\{S_1\}, \{(,)\}, \{S_1 \to (S_1)|S_1S_1|\epsilon\}, S)$$

*Gramáticas para esta linguaguem sem ϵ, em **FNC** e **FNG** são:*

$$G_{fnc} = (\{S, A, B, C\}, \{(,)\}, \{S \to AB \mid SS \mid AC, C \to SB, A \to (, B \to)\}, S)$$

$$G_{fng} = (\{R, X\}, \{(,)\}, \{R \to (X \mid (RX \mid (XR \mid (RXR\ ,\ X \to)\}, R)$$

Derivações pela esquerda para $()(())$:

$$\begin{aligned} S &\Rightarrow SS \Rightarrow ABS \Rightarrow (BS \Rightarrow ()S \Rightarrow ()AC \Rightarrow ()(C \\ &\Rightarrow ()(SB \Rightarrow ()(ABB \Rightarrow ()((BB \Rightarrow ()(()B \Rightarrow ()(()) \end{aligned}$$

$$\begin{aligned} R &\Rightarrow (XR \Rightarrow ()R \Rightarrow ()(RX \Rightarrow ()((XX \\ &\Rightarrow ()(()X \Rightarrow ()(()) \end{aligned}$$

6.5.1 Simplificações de gramáticas

Antes de considerar as transformações para as formas normais é necessário fazer algumas simplificações. Vamos ver que gramática que não gere ϵ é equivalente a outra que:

- não tenha variáveis *inúteis*, i.e que não gerem alguma sequência de terminais a partir do símbolo inicial
- não tenha produções-ϵ $A \to \epsilon$
- não tenha produções unitárias $A \to B$

Sem este tipo de regras é garantido que, em cada passo da derivação, a forma sentencial resultante:

- ou cresce estritamente
- ou contém mais um símbolo terminal

6.5.1.1 Eliminação de variáveis *inúteis*

Dado $G = (V, \Sigma, P, S)$, $X \in V$ é *útil* se gera alguma sequência de terminais a partir do símbolo inicial, i.e, se existe uma derivação $S \Rightarrow^\star \alpha X \beta \Rightarrow^\star w$ e $w \in \Sigma^\star$. Para que $X \in V$ seja útil:

1. $\exists w \in \Sigma^\star \; X \Rightarrow^\star w$, i.e,$X$ é um *gerador* de G
2. $\exists \alpha, \beta \in \Sigma \cup V^\star \; S \Rightarrow^\star \alpha X \beta$, i.e, X é *atingível* de S.

Lema 6.5. *Dada uma gramática independente de contexto $\mathcal{G} = (V, \Sigma, P, S)$ tal que $\mathcal{L}(\mathcal{G}) \neq \emptyset$, existe um algoritmo para determinar uma gramática independente de contexto $\mathcal{G}' = (V', \Sigma, P', S)$ equivalente a $\mathcal{G}$ em que cada variável $A \in V' \subseteq V$ gera algum $w \in \Sigma^\star$ (ou seja, $\forall A \in V' \exists w \in \Sigma^\star \; (A \Rightarrow^\star_{\mathcal{G}} w)$).*

Dem: A gramática $\mathcal{G}'$ pode ser obtida pelo método seguinte, o qual determina os conjuntos V' e P'. Começamos por considerar W como o conjunto vazio e V' como o conjunto $\{A \mid A \to \alpha \text{ para algum } \alpha \in \Sigma^\star\}$, e vamos completar V' pelo processo seguinte. Enquanto $W \neq V'$, fazemos consecutivamente as duas substituições seguintes: substituimos W por V'; e em seguida V' por $W \cup \{A \mid A \to \alpha \text{ para algum } \alpha \in (\Sigma \cup W)^\star\}$. Quando V' já não puder ser alterado, ou seja, quando $W = V'$, definimos P' como $\{A \to \alpha \mid A \in V', \; \alpha \in (\Sigma \cup V')^\star\}$. Note que as regras em P' são as regras em P que definem as variáveis em V', excluindo-se todas as regras que tenham, na parte direita, variáveis que não estão em V'. □

Exercício 6.10. *Mostra que em V' estão todos e só os* geradores *de G* ⋄

O algoritmo que acabámos de descrever pode ser aplicado a uma qualquer gramática $\mathcal{G}$ para verificar se a $\mathcal{L}(\mathcal{G}) = \{\}$: o símbolo inicial S da gramática dada não pertence ao conjunto V' das variáveis da gramática determinada se e só se $\mathcal{G}$ não gera qualquer palavra.

Exemplo 6.35. *Seja $\mathcal{G}$ a gramática dada por $\mathcal{G} = (\{S, C, D, M, R, Z, F, E, T\}, \{0, 1\}, P, S)$ onde P é constituído pelas regras seguintes.*

$$\begin{aligned} S &\to C \mid D \\ C &\to 0MC \mid 0RE \mid E \\ D &\to F \mid 0MD \mid 0RF \\ M &\to 0MM \mid 0RT \\ R &\to 0MR \mid 0RZ \mid 1 \\ Z &\to 1 \\ F &\to 1 \mid 1F \\ E &\to 1E \end{aligned}$$

De acordo com o método dado acima, começamos por considerar que $W := \emptyset$ e que $V' := \{Z, R, F\}$. Como $W \neq V'$, efectuamos as substituições descritas: $W := \{Z, R, F\}$ e $V' := \{Z, R, F\} \cup \{D\}$. Não se verifica $W = V'$. Temos que efectuar novamente substituições: vem $W := \{Z, R, F, D\}$ e $V' := \{Z, R, F, D\} \cup \{S\}$. Como ainda se tem $W \neq V'$, fazemos novamente as substituições, obtendo $W := \{Z, R, F, D, S\}$ e $V' := \{Z, R, F, D, S\} \cup \{\ \}$. Finalmente, W e V' são iguais. A gramática equivalente a $\mathcal{G}$ determinada pelo algoritmo descrito é

$$\mathcal{G}' = (\{Z, R, F, D, S\}, \{0, 1\}, P', S),$$

em que P' é o conjunto constituído pelas produções seguintes.

$$\begin{aligned} S &\to D \\ D &\to F \mid 0RF \\ R &\to 0RZ \mid 1 \\ Z &\to 1 \\ F &\to 1 \mid 1F \end{aligned}$$

Se aplicarmos a uma dada gramática $\mathcal{G} = (V, \Sigma, P, S)$ o algoritmo dado na prova do Lema **??**, não temos qualquer garantia de que sejam eliminados todos os símbolos (variáveis ou terminais) que não ocorrem em qualquer derivação a partir de S. Para isso, é necessário aplicar em seguida o algoritmo dado na prova do Lema **??**, que enunciamos abaixo.

Lema 6.6. *Dada uma gramática independente de contexto $\mathcal{G} = (V, \Sigma, P, S)$, existe um algoritmo que determina uma gramática independente de contexto $' = (V', \Sigma', P', S)$ equivalente a $\mathcal{G}$ em que cada variável $A \in V' \subseteq V$ é atíngivel de S (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 $(V' \cup \Sigma')^\star$).*

Dem: Como no caso anterior, V' e Σ' são os menores conjuntos tal que:

Base. $V' = \{S\}$ e $\Sigma' = \emptyset$

Indução. Se $X \in V'$ então todas as variáveis que ocorram em produções-X são colocados em V' e todos os terminais em Σ'

e $P' = P \cap \{X \to \alpha \mid X \in V'\}$

Proposição 6.8. *Seja $G = (V, \Sigma, P, S)$ uma g.i.c. e $L(G) \neq \emptyset$. Seja $G_1 = (V_1, \Sigma_1, P_1, S)$ obtida eliminando primeiro as variáveis não geradoras e depois, da gramática obtida, todos os símbolos não atíngiveis de S. Então G_1 não tem símbolos inúteis e $L(G) = L(G_1)$*

Dem: ($L(G_1) \subseteq L(G)$:) É evidente, porque apenas eliminamos produções.
($L(G) \subseteq L(G_1)$:) Se $S \Rightarrow_G^\star x$, todos os símbolos na derivação são geradores e atingíveis, portanto é uma derivação de G_1.

Exemplo 6.36. *Considerar a gramática $\mathcal{G} = (\{A, B, C, D\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, P, A)$ cujas produções são*

$$\begin{aligned} A &\to \mathtt{aa}D\mathtt{b} \mid B \\ B &\to \mathtt{b} \mid \mathtt{ba}B \\ D &\to \varepsilon \mid \mathtt{aa}D\mathtt{b} \\ C &\to \mathtt{c}CD \mid \mathtt{c} \end{aligned}$$

*Se aplicarmos a $\mathcal{G}$ o algoritmo dado na prova do Lema **??**, não conseguimos efectuar qualquer simplificação. Apliquemos agora o algoritmo dado na prova do Lema **??**, 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':=$\{A\}$, Σ':=$\{\ \}$ e P':=$\{\ \}$. Juntamos a P' todas as A-produções, ou seja P':= $\{A \to \mathtt{aa}D\mathtt{b}, A \to B\}$, a V' as variáveis B e D, e a Σ' os símbolos a *e* b.
Como alterámos V', juntamos a P' todas as B-produções e D-produções, o que não traz novas entradas em V' nem em Σ', concluindo que $\mathcal{G}' = (\{A, B, D\}, \{\mathtt{a}, \mathtt{b}\}, P', A)$ com

$$P' = \{A \to \mathtt{aa}D\mathtt{b},\ A \to B,\ B \to \mathtt{b},\ B \to \mathtt{ba}B,\ D \to \varepsilon,\ D \to \mathtt{aa}D\mathtt{b}\}.$$

Exercício 6.11. *Eliminar os símbolos inúteis de:*

$$\begin{aligned} S &\to AB \mid a \\ A &\to b \end{aligned}$$

◇

Resolução ??

B não é gerador, portanto fica $S \to a$ e $A \to b$. Mas A não é atingível. Logo a gramática, equivalente à dada, sem símbolos inúteis é $(\{S\}, \{a, b\}, \{S \to a\}, S)$.

6.5.1.2 Eliminar produções-ϵ e produções unitárias

Lema 6.7. *Para cada $G = (V, \Sigma, P, S)$, existe uma gramática G' sem produções-ϵ nem produções unitárias e tal que $L(G') = L(G) \setminus \{\epsilon\}$.*

Demonstração. Seja $\widehat{P}$ o menor conjunto que contém P e é fechado para as seguintes regras:

(a) se $A \to \alpha B \beta$ e $B \to \epsilon$ estão em $\widehat{P}$, então $A \to \alpha\beta \in \widehat{P}$

(b) se $A \to B$ e $B \to \gamma$ estão em $\widehat{P}$, então $A \to \gamma \in \widehat{P}$

Nota que este processo é finito assim como o $\widehat{P}$!
Seja $\widehat{G} = (V, \Sigma, \widehat{P}, S)$. Como $P \subseteq \widehat{P}$, temos $L(G) \subseteq L(\widehat{G})$. Mas ,temos mesmo, $L(G) = L(\widehat{G})$, porque cada produção acrescentada correspondia a dois passos de derivação em G.
Vamos ver que para $x \in \Sigma^\star \setminus \{\epsilon\}$, qualquer derivação de tamanho mínimo $S \Rightarrow^\star_{\widehat{G}} x$ não usa nenhuma produções-ϵ nem produções unitárias.
Logo poderão ser apagadas de $\widehat{P}$. G' é a gramática que se obtém de $\widehat{G}$ eliminando essas produções.
Seja $x \neq \epsilon$ e seja a derivação de menor tamanho $S \Rightarrow^\star_{\widehat{G}} x$.
Suponhamos, por contradição, que uma produção-ϵ é usada:
$S \Rightarrow^\star_{\widehat{G}} \gamma B \delta \Rightarrow^1_{\widehat{G}} \gamma\delta \Rightarrow^\star_{\widehat{G}} x$.
Mas $\gamma\delta$ tem de ser não nulo (senão $x = \epsilon$) e portanto tem de haver uma forma sentencial anterior que já continha B:
$S \Rightarrow^m_{\widehat{G}} \eta A \theta \Rightarrow^1_{\widehat{G}} \eta\alpha B\beta\theta \Rightarrow^n_{\widehat{G}} \gamma B\delta \Rightarrow^1_{\widehat{G}} \gamma\delta \Rightarrow^k_{\widehat{G}} x$
mas, então, por **??**, $A \to \alpha\beta \in \widehat{P}$ e portanto podia ter sido aplicada, conduzindo a uma derivação de x de tamanho menor:
$S \Rightarrow^m_{\widehat{G}} \eta A \theta \Rightarrow^1_{\widehat{G}} \eta\alpha\beta\theta \Rightarrow^n_{\widehat{G}} \gamma\delta \Rightarrow^k_{\widehat{G}} x$
Absurdo!
De modo similar, podemos mostrar que produções unitárias não aparecem em derivações de tamanho mínimo.
Seja $x \neq \epsilon$ e seja a derivação de menor tamanho $S \Rightarrow^\star_{\widehat{G}} x$.
Suponhamos, por contradição, que uma produção unitária $A \to B$ é usada:
$S \Rightarrow^\star_{\widehat{G}} \alpha A \beta \Rightarrow^1_{\widehat{G}} \alpha B \beta \Rightarrow^\star_{\widehat{G}} x$
mas então, mais tarde B será substituído:
$S \Rightarrow^m_{\widehat{G}} \alpha\alpha A\beta \Rightarrow^1_{\widehat{G}} \alpha B\beta \Rightarrow^n_{\widehat{G}} \eta B\theta \Rightarrow^1_{\widehat{G}} \eta\gamma\theta \Rightarrow^k_{\widehat{G}} x$
Mas, então, por **??**, $A \to \gamma \in \widehat{P}$ e portanto podia ter sido aplicada, conduzindo a uma derivação de x de tamanho menor:
$S \Rightarrow^m_{\widehat{G}} \alpha A \beta \Rightarrow^1_{\widehat{G}} \alpha\gamma\beta \Rightarrow^n_{\widehat{G}} \eta\gamma\theta \Rightarrow^k_{\widehat{G}} x$
Absurdo! □

Exemplo 6.37. *Vamos aplicar o processo descrito na prova anterior para determinar 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 $T \to \varepsilon$, marcamos T como variável que gera ε. Consequentemente, como $S \to T$, também S é marcado. Como S e T estão marcados e $C \to ST$, 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, a regra $S \to T$ é incluída em P', não sendo incluída a regra $S \to \varepsilon$ resultante da substituição de T por ε (como referido na prova do lema).
A gramática $\mathcal{G}'$ dada por $\mathcal{G}' = (\{S, T, C\}, \{\mathtt{a}, \mathtt{b}\}, P', S)$, cujas regras em P' são

$$\begin{aligned} T &\to \mathtt{a} \mid \mathtt{a}T \\ S &\to T \mid \mathtt{a}S\mathtt{b} \mid \mathtt{ab} \\ C &\to S \mid T \mid ST \mid \mathtt{ab} \end{aligned}$$

gera $\mathcal{L}(\mathcal{G}) \setminus \{\varepsilon\}$ e não tem variáveis que gerem ε. Note que todas as variáveis geram sequências de terminais. Assim, se aplicarmos a $\mathcal{G}'$ o algoritmo dado no Lema **??**, *não conseguimos efectuar qualquer simplificação. No entanto, há variáveis (mais precisamente C) que não ocorrem em derivações a partir de S (símbolo inicial). Consequentemente, se aplicarmos o algoritmo descrito no Lema* **??**, *conseguimos eliminar C, obtendo 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:*

$$(\{S, T\}, \{\mathtt{a}, \mathtt{b}\}, \{T \to \mathtt{a},\ T \to \mathtt{a}T,\ S \to T,\ S \to \mathtt{a}S\mathtt{b},\ S \to \mathtt{ab}\}).$$

Exemplo 6.38. *Seja $\mathcal{G}$ a gramática dada por*

$$\mathcal{G} = (\{S, T\}, \{\mathtt{a}, \mathtt{b}\}, \{T \to \mathtt{a},\ T \to \mathtt{a}T,\ S \to T,\ S \to \mathtt{a}S\mathtt{b},\ S \to \mathtt{ab}\}).$$

Segundo a prova do Lema **??**, *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 $A \to \varepsilon$.*

Exercício 6.12. *Eliminar os símbolos inúteis, as produções-ϵ e as produções unitárias da seguinte gramática:*

$$\begin{array}{lcll} S & \to & ASB & | \ \epsilon \\ A & \to & aAS & | \ a \\ B & \to & SbS & | \ A \ | \ bb \end{array}$$

◇

Resolução ??

A gramática não tem símbolos inúteis. Como $S \to \epsilon$, acrescentam-se as produções $S \to AB$, $A \to aA$, $B \to Sb$, $B \to bS$, $B \to b$, $B \to a$, $B \to aAS$, $B \to aA$. Eliminando as produções-ϵ e as produções unitárias, vem:

$$\begin{array}{lcllllllll} S & \to & ASB & | \, AB & & & & & & \\ A & \to & aAS & | \, aA & | \, a & & & & & \\ B & \to & SbS & | \, Sb & | \, bS & | \, b & | \, a & | \, aA & | \, aAS & | \, bb \end{array}$$

6.5.2 Forma Normal de Chomsky

Vamos agora ver como converter uma g.i.c. $G = (V, \Sigma, P, S)$, com $\epsilon \notin L(G) \neq \emptyset$ numa equivalente em FNC. Depois de eliminarmos as produções-ϵ, as produções unitárias, e as variávies inúteis ou se tem produções da forma $A \to a$ ou de tamanho ≥ 2. Para obter a FNC:

1. Para cada $a \in \Sigma$ introduz-se um não-terminal A_a e a produção $A_a \to a$. E substitui-se cada ocorrência de a nos lados direitos de outras regras por A_a.
2. Então, todas as produções são da forma $A \to a$ ou $A \to B_1 B_2 \dots B_k$, $k \geq 2$ e $B_i \in V$.
3. Para cada produção $A \to B_1 B_2 \dots B_k$, $k \geq 3$, introduz-se um novo não terminal C e substituí-se essa produção por duas novas: $A \to B_1 C$ e $C \to B_2 \dots B_k$ Continua-se este passo até não haver mais regras com lados direitos de tamanho maior que 2.

Exemplo 6.39. *Dada gramática para a linguagem L de parêntesis "bem casados" $G = (\{S\}, \{(,)\}, \{S \to (S)|SS|\epsilon\}, S)$ vamos determinar uma FNC para $L(G) \setminus \{\epsilon\}$:*

- *Eliminando as produções-ϵ e unitárias vem: $S \to (S)|SS|()$*
- *Não tem não terminais inúteis*
- *Introduzimos dois não terminais A e B, e substituímos as produções:*

  $S \to ASB|SS|AB, \quad A \to (\quad B \to)$
- *Finalmente, adiciona-se um novo não terminal e substituí-se $S \to ASB$ por $S \to AC$ e $C \to SB$. A gramática obtida é a dada na página **??**.*

Proposição 6.9. *Qualquer linguagem independente de contexto, que não tenha ε, é gerada por uma gramática na forma normal de Chomsky.*

Dem: 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.
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 $A \to X_1 X_2 \cdots X_n$ 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 $A \to X_1 X_2 \cdots X_n$, com $n \geq 2$, por regras da forma $A \to Y_1 Y_2 \cdots Y_n$, 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 está 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 $D_1, \ldots D_{n-2}$ e as regras $A \to Y_1 D_1$, $D_1 \to Y_2 D_2$, $\ldots$, e $D_{n-2} \to Y_{n-1} Y_n$.
A gramática resultante está na forma normal de Chomsky e pode verificar-se que é equivalente à dada. □

Exemplo 6.40. *Seja* $\mathcal{G} = (\{S, T\}, \{\mathtt{a}, \mathtt{b}\}, \{T \to \mathtt{a},\ T \to \mathtt{a}T,\ S \to \mathtt{a},\ S \to \mathtt{a}T,\ S \to \mathtt{a}S\mathtt{b},\ S \to \mathtt{ab}\})$. *Para obter uma gramática na forma normal de Chomsky que seja equivalente a* $\mathcal{G}$*, vamos aplicar o método que acábamos de descrever. 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 seguinte, a qual gera* $\mathcal{L}(\mathcal{G})$ *e está na forma normal de Chomsky, sendo* $V = \{S, T, A, B, X\}$*.*

$$(V, \{\mathtt{a}, \mathtt{b}\}, \{T \to \mathtt{a}, T \to AT,\ A \to \mathtt{a},\ S \to \mathtt{a},\ S \to AT,\ S \to AX,\ X \to SB,\ B \to \mathtt{b},\ S \to AB\}, S)$$

Exemplo 6.41. *Seja* $\mathcal{G} = (\{E, F, T, N, D, R\}, \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}, \mathtt{(}, \mathtt{)}, \mathtt{+}, \mathtt{-}, \mathtt{*}, \mathtt{/}\}, P, E)$ *onde*

P é constituído pelas regras seguintes,

$$\begin{aligned}
E &\rightarrow T\texttt{+}E \mid T\texttt{-}E \mid T \\
T &\rightarrow F \mid F\texttt{*}T \mid F\texttt{/}T \\
F &\rightarrow \texttt{(}E\texttt{)} \mid N \\
N &\rightarrow \texttt{0} \mid D \mid DR \\
D &\rightarrow \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
R &\rightarrow \texttt{0} \mid D \mid \texttt{0}R \mid DR
\end{aligned}$$

a qual gera algumas das expressões aritméticas envolvendo inteiros não negativos representados em decimal (isto é, na base 10).
Determinemos uma gramática na forma de Chomsky equivalente a $\mathcal{G}$. Se analisarmos as regras da gramática, vemos que não há variáveis que produzam ε, mas há produções unitárias. Apliquemos o método descrito na prova do Lema **??** *para determinar uma gramática*

$$\mathcal{G}' = (\{E,F,T,N,D,R\}, \{\texttt{0},\texttt{1},\texttt{2},\texttt{3},\texttt{4},\texttt{5},\texttt{6},\texttt{7},\texttt{8},\texttt{9},\texttt{(},\texttt{)},\texttt{+},\texttt{-},\texttt{*},\texttt{/}\}, P', E)$$

equivalente a $\mathcal{G}$, mas sem regras unitárias. Todas as regras não unitárias de $\mathcal{G}$ fazem parte do conjunto P', pelo que temos

$$\begin{aligned}
E &\rightarrow T\texttt{+}E \mid T\texttt{-}E \\
T &\rightarrow F\texttt{*}T \mid F\texttt{/}T \\
F &\rightarrow \texttt{(}E\texttt{)} \\
N &\rightarrow \texttt{0} \mid DR \\
D &\rightarrow \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
R &\rightarrow \texttt{0} \mid \texttt{0}R \mid DR
\end{aligned}$$

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

$$E \rightarrow F\texttt{*}T \mid F\texttt{/}T \mid \texttt{(}E\texttt{)} \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

e como $T \Rightarrow^\star_{\mathcal{G}} F$, $T \Rightarrow^\star_{\mathcal{G}} N$ e $T \Rightarrow^\star_{\mathcal{G}} D$, introduzimos em P'

$$T \rightarrow \texttt{(}E\texttt{)} \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

e por $F \Rightarrow_{\mathcal{G}}^{\star} N$ *e* $F \Rightarrow_{\mathcal{G}}^{\star} D$*, introduzimos em* P' *as regras*

$$F \rightarrow \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

e por $N \Rightarrow_{\mathcal{G}}^{\star} D$*, introduzimos ainda*

$$N \rightarrow \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

e finalmente, por $R \Rightarrow_{\mathcal{G}}^{\star} D$*, acrescentamos*

$$R \rightarrow \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

podendo assim eliminar todas as produções unitárias.
Para determinar uma gramática equivalente mas que esteja na forma de Chomsky, aplicamos a seguir o algoritmo descrito na prova da Proposição **??**. *Não alteramos as regras cuja parte direita seja um único terminal. Nas restantes, substituimos cada terminal por uma nova variável.*

$M \rightarrow \texttt{+}$

$S \rightarrow \texttt{-}$

$X \rightarrow \texttt{*}$

$Q \rightarrow \texttt{/}$

$A \rightarrow \texttt{(}$

$B \rightarrow \texttt{)}$

$O \rightarrow \texttt{0}$

$E \rightarrow TME \mid TSE \mid FXT \mid FQT \mid AEB \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$

$T \rightarrow FXT \mid FQT \mid AEB \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$

$F \rightarrow AEB \mid \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$

$N \rightarrow \texttt{0} \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$

$D \rightarrow \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$

$R \rightarrow \texttt{0} \mid OR \mid DR \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$

Para chegar à forma de Chomsky resta substituir as regras cuja parte direita tenha três ou mais variáveis. Substituimos $E \to TME \mid TSE \mid FXT \mid FQT \mid AEB$ *por*

$$\begin{aligned} E &\to TE_1 \mid TE_2 \mid FT_1 \mid FT_2 \mid AT_3 \\ E_1 &\to ME \\ E_2 &\to SE \\ T_1 &\to XT \\ T_2 &\to QT \\ T_3 &\to EB \end{aligned}$$

a para obter uma gramática mais simples, substituimos $T \to FXT \mid FQT \mid AEB$ *por*

$$T \to FT_1 \mid FT_2 \mid AT_3$$

aproveitando as variáveis anteriores, e finalmente substituimos a regra $F \to AEB$ *por*

$$F \to AT_3$$

em vez de a substituirmos por $F \to AF_1$ *e* $F_1 \to EB$

Bibliografia