Linguagens Formais e Autômatos

Marcus Vinícius Midena Ramos

Curso de Engenharia de Computação
Universidade Federal do Vale do São Francisco

22 de abril de 2008

Sumário

1 Elementos de Matemática Discreta

As **linguagens formais** (ou linguagens estruturadas em frases) podem ser vistas como conjuntos. Conseqüentemente, muito da teoria e dos principais resultados da área de linguagens formais está baseado na ainda mais fundamental teoria dos conjuntos da matemática discreta.

A teoria dos conjuntos é relativamente extensa, e dela serão apresentados neste capítulo apenas os tópicos, conceitos e definições que se mostram mais importantes para a fundamentação e o estudo dos capítulos seguintes.

1.1 Conjuntos

Um **conjunto** é uma coleção de símbolos, também denominados átomos ou elementos, em que não são consideradas ocorrências múltiplas dos mesmos nem há relação de ordem entre eles.

Exemplo 1.1 A inclusão do símbolo $\diamondsuit$ no conjunto $\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$ resulta no próprio conjunto $\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$, pois o mesmo já faz parte do conjunto e, portanto, não deve ser considerado novamente. Por outro lado, o conjunto $\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$ é igual ao conjunto $\{\diamondsuit, \clubsuit, \spadesuit, \heartsuit\}$, uma vez que não existe relação de ordem entre os elementos que os compõem. □

Um **símbolo** corresponde a uma representação gráfica única e indivisível. Se formado por caracteres, um símbolo pode ser composto por um número arbitrário deles.

Exemplo 1.2 São exemplos de símbolos: "a", "abc", "$\spadesuit$", "1" etc. □

Alguns conjuntos podem ser especificados através da simples **enumeração** de todos os seus elementos, denotados entre chaves e separados por vírgulas.

Exemplo 1.3 O conjunto formado pelos elementos $0, 1, 2, 3$ é representado por $\{0, 1, 2, 3\}$. O conjunto $\{a, b, c, d, e, f\}$ é formado pelas seis primeiras letras do alfabeto romano. O conjunto $\{01, 231, 33, 21323\}$ contém os elementos $01, 231, 33$ e 21323. □

Conjuntos podem ser referenciados através de nomes, arbitrariamente escolhidos.

Exemplo 1.4 $X = \{0, 1, 2, 3\}$, $Y = \{a, b, c, d, e, f\}$. Assim, os nomes X e Y passam a denotar os conjuntos correspondentes. □

O número de elementos contido em um conjunto A é denotado por $\big|A\big|$.

Exemplo 1.5 No exemplo 1.4, $\big|X\big| = 4$, $\big|Y\big| = 6$. □

Os símbolos $\in$ e $\notin$ servem para denotar se um determinado elemento **pertence** ou **não pertence** a um conjunto, respectivamente.

Exemplo 1.6 No exemplo 1.4, $0 \in X$, $5 \notin X$, $2 \notin Y$, $b \notin X$, $c \in Y$, $h \notin Y$. □

Conjuntos podem conter um número finito ou infinito de elementos. No primeiro caso, o conjunto pode ser denotado enumerando-se (relacionando-se explicitamente) todos os elementos que o compõem, como foi feito para os conjuntos X e Y do exemplo 1.4, que são **conjuntos finitos**.

Conjuntos infinitos podem ser denotados através da especificação (formal ou informal) de regras ou propriedades que devem ser satisfeitas por todos os seus elementos, possibilitando assim a sua identificação precisa e completa a partir de uma especificação finita.

Exemplo 1.7 $P = \{x \mid x \text{ é um número primo}\}$, $Q = \{y \mid \exists\ n \text{ inteiro tal que } y = n^2\}$. O primeiro exemplo deve ser lido da seguinte forma: "P é o conjunto formado pelos elementos x, tal que x é um número primo". Em outras palavras, P é o conjunto, infinito, formado por todos os números primos: $\{1, 2, 3, 5, 7, 11, 13, 17...\}$. O conjunto Q, também infinito, é formado por todos os números que correspondem ao quadrado de algum número inteiro: $\{0, 1, 4, 9, 16...\}$. □

Quando um conjunto é especificado a partir de regras, o símbolo "$\mid$" deve ser lido como "tal que", e serve para introduzir as condições que devem ser satisfeitas pelos membros do conjunto, que assim tornam-se conhecidos.

O conjunto que não contém nenhum elemento recebe o nome de **conjunto vazio**. Por definição, $|\varnothing| = 0$. O conjunto vazio é denotado por $\varnothing$ ou ainda pelo símbolo $\{\}$. Assim, $\{\} = \varnothing$.

Dois conjuntos são ditos **idênticos**, ou simplesmente **iguais**, se eles contêm exatamente os mesmos elementos. A igualdade de dois conjuntos é denotada através do símbolo "=".

Exemplo 1.8 Considere $Z = \{a, b\}$ e $W = \{b, a\}$. Então, $Z = W$. □

Um conjunto A é dito "**contido** em um conjunto B", condição esta denotada através do símbolo "$\subseteq$", se todo elemento de A for também elemento de B. Neste caso diz-se, equivalentemente, que "A é um **subconjunto** de B" ou, ainda, que "B **contém** A". Os conjuntos $\varnothing$ e A são, por definição, subconjuntos de qualquer conjunto A.

Exemplo 1.9 Para os conjuntos $A = \{b, c, d\}$, $B = \{a, b, c, d, e\}$ e $C = \{e, a, d, b, c\}$ tem-se que $A \subseteq B$ e $B \subseteq C$. Portanto, pode-se dizer que A está contido em B e em C, que A é subconjunto de B e de C, que C contém A e B e, ainda, que B e C são subconjuntos um do outro ou que estão contidos um no outro. B e C, por outro lado, não estão contidos em A. □

Dois conjuntos M e N são iguais se e somente se $M \subseteq N$ e $N \subseteq M$, e tal igualdade é denotada por $M = N$. A **desigualdade** de dois conjuntos é expressa através do símbolo "$\neq$", ocorrendo portanto quando no máximo apenas uma das duas condições $M \subseteq N$ e $N \subseteq M$ for verdadeira.

Exemplo 1.10 No exemplo 1.9, $A \subseteq B$, porém $A \neq B$. Como $B \subseteq C$ e $C \subseteq B$, então $B = C$. □

Se $M \subseteq N$ e $M \neq N$, diz-se que M é um **subconjunto próprio** de N. O símbolo $\subset$ denota essa condição: $M \subset N$. O conjunto $\varnothing$ é subconjunto próprio de qualquer conjunto, exceto do próprio conjunto $\varnothing$.

Exemplo 1.11 No exemplo 1.9, A é subconjunto próprio de B, porém B não é subconjunto próprio de C. Logo, $A \subset B$. □

Algumas operações importantes sobre conjuntos são apresentadas a seguir.

Conjunto potência ("**powerset**"): Denotado por 2^A, onde A é um conjunto. Essa operação é utilizada para designar o conjunto formado por todos os possíveis subconjuntos de A:

$$2^A = \{B \mid B \subseteq A\}$$

Para conjuntos A finitos, $\left|2^A\right| = 2^{|A|}$.

Exemplo 1.12 Para $A = \{0, 1, 2\}$, temos $2^A = \{\varnothing, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}$. Além disso, $|A| = 3$ e $\left|2^A\right| = 2^3 = 8$. □

União: A união de dois conjuntos A e B corresponde ao conjunto formado por todos os elementos contidos em cada um dos dois conjuntos A e B. Elementos repetidos em ambos os conjuntos são considerados uma única vez no conjunto união:

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

Trata-se de uma operação associativa, ou seja, uma operação para a qual vale a propriedade:

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

A generalização da operação de união é denotada da seguinte forma:

$$\bigcup_{i=0}^{n} A_i = A_0 \cup A_1 \cup A_2 \cup ... \cup A_n$$

Exemplo 1.13 $\{a, b, c\} \cup \{c, d\} = \{a, b, c, d\}$. $\{a, b, c, d\} \cup \varnothing = \{a, b, c, d\}$. □

Intersecção: Define-se a intersecção de dois conjuntos A e B como sendo a coleção de todos os elementos comuns aos dois conjuntos:

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

Também em decorrência da associatividade desta operação, a sua generalização é denotada de forma similar ao caso da união:

$$\bigcap_{i=0}^{n} A_i = A_0 \cap A_1 \cap A_2 \cap ... \cap A_n$$

Exemplo 1.14 $\{a, b, c\} \cap \{c, d\} = \{c\}$. $\{a, b\} \cap \{c, d\} = \varnothing$. $\{a, b, c, d\} \cap \varnothing = \varnothing$. □

Dois conjuntos A e B são ditos **disjuntos** se $A \cap B = \varnothing$.

Exemplo 1.15 Os conjuntos $\{a, b, c\}$ e $\{c, d\}$ não são disjuntos, pois $\{a, b, c\} \cap \{c, d\} = \{c\} \neq \{\}$. Os conjuntos $\{a, b\}$ e $\{c, d\}$ são disjuntos, pois $\{a, b\} \cap \{c, d\} = \varnothing$. □

Diferença: Define-se a diferença entre dois conjuntos A e B (nesta ordem) como sendo o conjunto formado por todos os elementos de A não-pertencentes ao conjunto B. Denota-se este conjunto como:

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

Exemplo 1.16 $\{a, b, c\} - \{c, d\} = \{a, b\}$. $\{a, b\} - \{a, b, c\} = \varnothing$. $\{a, b, c\} - \{d, e\} = \{a, b, c\}$. $\{c, d\} - \{a, b, c\} = \{d\}$. $\{a, b, c\} - \{a, b\} = \{c\}$. $\{d, e\} - \{a, b, c\} = \{d, e\}$. □

Complementação: Define-se a complementação de um conjunto A em relação ao conjunto B, $A \subseteq B$, como sendo o conjunto de todos os elementos de B que não pertencem a A. Denota-se este conjunto como:

$$\overline{A}_B = B - A$$

Muitas vezes esta operação é definida para um conjunto A em relação a um outro conjunto B subentendido e, neste caso, escreve-se simplesmente:

$$\overline{A} = B - A$$

Diz-se, neste caso, que o conjunto subentendido é o conjunto universo da operação. O resultado da operação é conhecido simplesmente como **complemento** de A.

Exemplo 1.17 Sejam $A = \{a, b, c\}$, $B = \{a, b, c, d\}$ e $C = \{d, c, a, b\}$. Então, $\overline{A}_B = \{d\}$ e $\overline{B}_C = \varnothing$. Sendo $D = \{a, b, c, d, e\}$ o conjunto universo, $\overline{A} = \{d, e\}, \overline{B} = \overline{C} = \{e\}$ e $\overline{D} = \varnothing$. □

Produto cartesiano: O produto cartesiano de dois conjuntos é o conjunto formado por todos os pares ordenados (a, b), em que a é um elemento de A, e b um elemento de B:

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

Um **par ordenado** é uma representação de dois elementos separados por vírgula e delimitados por parênteses, como em (a, b). Tal representação implica uma relação de ordem em que o elemento a é anterior ao elemento b. Conseqüentemente, se $a \neq b$, então $(a, b) \neq (b, a)$.

Se A e B são conjuntos finitos, então $\big|A \times B\big| = \big|A\big| * \big|B\big|$.

A generalização desta operação é denotada:

$$A_1 \times A_2 \times ... \times A_n = \{(a_1, a_2, ..., a_n) \mid a_i \in A_i \text{ para } 1 \leqslant i \leqslant n\}$$

Exemplo 1.18 Sejam $A = \{a, b, c\}$ e $B = \{0, 1\}$. Então $A \times B =$

$$\{(a, 0), (a, 1), (b, 0), (b, 1), (c, 0), (c, 1)\}$$

e

$$\big|A \times B\big| = \big|A\big| * \big|B\big| = 3 * 2 = 6$$

□

Partição: Define-se partição de um conjunto A como sendo qualquer coleção formada por n subconjuntos não-vazios de A, $n \geq 1$, tal que:

$$A = \bigcup_{i=0}^{n} A_i \quad \text{e} \quad \bigcup_{i=0}^{n} \left(\bigcup_{j=0}^{n, j \neq i} A_i \cap A_j \right) = \varnothing$$

Exemplo 1.19 Seja $A = \{a, b, c, d\}$. Então, $\{\{a, b\}, \{c, d\}\}$ é uma partição de A. Da mesma forma, o conjunto $\{\{a\}, \{b\}, \{c\}, \{d\}\}$, bem como $\{\{a, b, c, d\}\}$, entre vários outros. □

A seguir serão apresentados três importantes resultados acerca de conjuntos, os dois primeiros conhecidos como **Leis de De Morgan**, os quais serão úteis na demonstração de outros teoremas mais adiante no texto.

Teorema 1.1 (Leis de De Morgan) *Sejam A e B dois conjuntos quaisquer. Então $A \cap B = \overline{\overline{A} \cup \overline{B}}$ e $A \cup B = \overline{\overline{A} \cap \overline{B}}$.*

Justificativa Estas propriedades podem ser inferidas, respectivamente, pela inspeção dos diagramas das Figuras 1.1 e 1.2.

Na Figura 1.1, da esquerda para a direita, as áreas hachuradas dos diagramas representam, respectivamente, $\overline{A}, \overline{B}, \overline{A} \cup \overline{B}$ e $\overline{\overline{A} \cup \overline{B}}$.

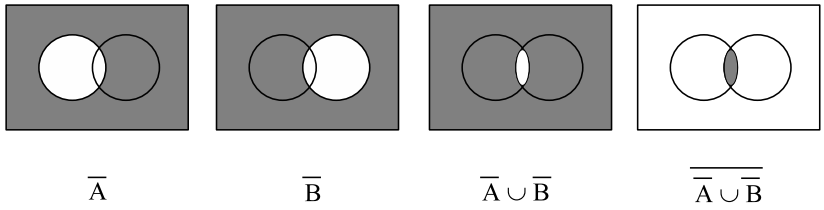

Figura 1.1: Demonstração da Lei de De Morgan para intersecção de conjuntos

Na Figura 1.2, da esquerda para a direita, as áreas hachuradas dos diagramas representam, respectivamente, $\overline{A}, \overline{B}, \overline{A} \cap \overline{B}$ e $\overline{\overline{A} \cap \overline{B}}$.

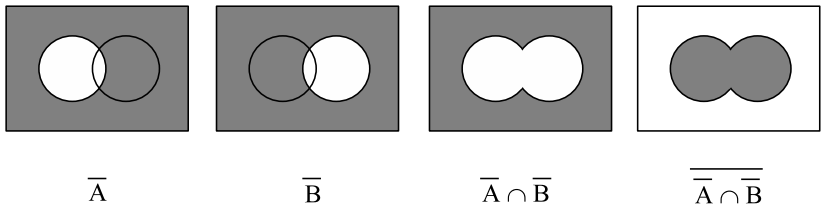

Figura 1.2: Demonstração da Lei de De Morgan para união de conjuntos

■

Teorema 1.2 (Igualdade de conjuntos) *Sejam A e B dois conjuntos quaisquer. Então $A = B \Leftrightarrow (A \cap \overline{B}) \cup (\overline{A} \cap B) = \varnothing$.*

Justificativa
($\Rightarrow$) Se $A = B$, então $(A \cap \overline{B}) \cup (\overline{A} \cap B) = (A \cap \overline{A}) \cup (\overline{A} \cap A) = \varnothing \cup \varnothing = \varnothing$.
($\Leftarrow$) Se $(A \cap \overline{B}) \cup (\overline{A} \cap B) = \varnothing$, então as duas seguintes condições devem ser simultaneamente satisfeitas:

1. $(A \cap \overline{B}) = \varnothing$;
2. $(\overline{A} \cap B) = \varnothing$.

Considere-se $A \subseteq C$ e $B \subseteq C$, de forma que $\overline{A} = \overline{A}_C$ e $\overline{B} = \overline{B}_C$. Então, existem apenas três possibilidades para representar a relação entre A e B:

i $A \neq B$ e $A \cap B \neq \varnothing$. Logo, $A \cap \overline{B} \neq \varnothing$;

ii $A \neq B$ e $A \cap B = \varnothing$. Logo, $A \cap \overline{B} \neq \varnothing$;

iii $A = B$. Logo, $A \cap \overline{B} = \varnothing$.

Portanto, a única relação possível entre A e B que satisfaz à condição (1) é a relação (iii). Da mesma forma, pode-se facilmente mostrar que (iii) também é a única relação que satisfaz à condição (2), e isso completa a demonstração do teorema. ■

A menos de ressalva em contrário, ao longo deste texto os nomes de conjuntos serão representados por intermédio das letras maiúsculas do alfabeto romano (A, B, X, Y etc.). Elementos de um conjunto são usualmente denotados através das letras minúsculas do mesmo alfabeto (a, b, c etc.).

Os seguintes conjuntos serão utilizados no restante deste livro:

- $\mathbb{N}$, representando os números naturais $\{0, 1, 2, 3, ...\}$;
- $\mathbb{Z}$, representando os números inteiros $\{..., -3, -2, -1, 0, 1, 2, 3, ...\}$;
- $\mathbb{Z}_+$, representando os números inteiros positivos $\{1, 2, 3, ...\}$;
- $\mathbb{Z}_-$, representando os números inteiros negativos $\{..., -3, -2, -1\}$;
- $\mathbb{R}$, representando os números reais.

1.2 Relações

Uma **relação** R sobre dois conjuntos A e B é definida como um subconjunto de $A \times B$.

Relações representam abstrações de conceitos matemáticos fundamentais, como, por exemplo, as operações aritméticas, lógicas e relacionais, além de constituírem a base teórica para o estudo sistemático das funções. O conjunto de todas as relações definíveis sobre $A \times B$ é dado por $2^{A \times B}$.

Exemplo 1.20 A relação $R_1 = \{(a, b) \mid a, b \in \mathbb{N} \text{ e } a > b\}$, sobre $\mathbb{N} \times \mathbb{N}$, contém, entre infinitos outros, os elementos $(2, 1), (7, 4)$ e $(9, 3)$. A relação $R_2 = \{(x, y, z) \mid x, y, z \in \mathbb{Z} \text{ e } x^2 = y^2 + z^2\}$, sobre $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$, contém os elementos $(0, 0, 0), (1, 1, 1), (-1, -1, -1), (5, 4, 3), (-10, 8, -6)$ etc. □

Uma relação R aplicada sobre um elemento a de um conjunto A e outro elemento b de um conjunto B pode ser denotada, em notação infixa, por aRb. Se $(a, b) \in R$, diz-se, de forma abreviada, que aRb.

Os conjuntos A e B recebem, respectivamente, os nomes **domínio** e **co-domínio** (ou **contradomínio**) da relação R. Por envolver dois conjuntos, essa relação é dita **binária** e seus elementos recebem a designação de **pares ordenados**. Relações binárias sobre um mesmo conjunto A representam subconjuntos de $A \times A$.

Exemplo 1.21 Considere-se a relação binária "$\neq$" sobre o conjunto dos números inteiros. Essa relação se define como o conjunto dos pares ordenados tais que suas duas componentes são diferentes. Alguns dos elementos do conjunto definido por essa relação são $(1, 3), (-5, 0), (8, -2)$ etc. Utilizando a notação introduzida, os elementos citados, pertencentes a essa relação, são denotados por $1 \neq 3$, $-5 \neq 0$ e $8 \neq -2$, coincidindo, portanto, com a representação tradicional da relação.

Notar que $(1, 1), (0, 0)$ e $(-5, -5)$ são exemplos de pares ordenados que não satisfazem a essa relação binária, pois suas duas componentes coincidem. □

O conceito de relação pode ser generalizado para mais de dois conjuntos, consistindo, sempre, em subconjuntos definidos sobre o produto cartesiano dos conjuntos participantes da relação. A relação, nesse caso, é dita uma relação "n-ária", e corresponde a um subconjunto do produto cartesiano dos conjuntos envolvidos. Sejam n conjuntos $A_1, A_2, ...A_n$. Os elementos pertencentes ao conjunto definido por uma relação n-ária sobre $A_1, A_2, ...A_n$ são, portanto, elementos de $A_1 \times A_2 \times ... \times A_n$, e têm a seguinte forma:

$$(a_1, a_2, a_3, ..., a_n)$$

onde $a_1 \in A_1, a_2 \in A_2, ...a_n \in A_n$.

Tais elementos são denominados **ênuplas ordenadas**. Em casos particulares, como para $n = 2, 3, 4, 5$ etc., as ênuplas recebem nomes especiais, geralmente os ordinais de n: pares, triplas, quádruplas, quíntuplas etc. Quando n é grande, usa-se em geral o nome "n-tupla ordenada". Por exemplo, $(a_1, a_2, ...a_{10})$ é considerada uma décupla (ou uma 10-tupla) ordenada.

Uma relação binária R sobre um conjunto A é dita:

- **Reflexiva**: se $aRa, \forall\, a \in A$;
- **Simétrica**: se aRb implica $bRa, \forall\, a, b \in A$;
- **Transitiva**: se aRb e bRc implicam $aRc, \forall\, a, b, c \in A$;

sendo que a, b, c não precisam ser necessariamente distintos.

Exemplo 1.22 A relação binária "identidade" (=) definida sobre o conjunto dos números inteiros $\mathbb{Z}$ como o conjunto de todos os pares ordenados para os quais as duas componentes são idênticas. Ela é reflexiva, pois $a = a, \forall\, a \in \mathbb{Z}$; é simétrica, pois $a = b$ implica $b = a, \forall\, a, b \in \mathbb{Z}$; e transitiva, uma vez que $a = b$ e $b = c$ implica $a = c, \forall\, a, b, c \in \mathbb{Z}$. Alguns elementos do conjunto definido por essa relação são $(4, 4), (0, 0), (-7, -7)$ etc. Notar que pares ordenados, tais como $(1, -3), (0, 5)$ e $(7, 9)$, não pertencem a essa relação. □

Por outro lado, a relação binária "maior" ($>$), definida como o conjunto dos pares ordenados cujas primeiras componentes tenham valor maior que as segundas componentes, aplicada sobre o mesmo conjunto $\mathbb{Z}$, revela-se não-reflexiva, pois não é verdade que $a > a, \forall\, a \in \mathbb{Z}$; não-simétrica, já que $a > b$ não implica $b > a, \forall\, a$ e $b \in \mathbb{Z}$; porém ela é transitiva, uma vez que $a > b$ e $b > c$ implica $a > c, \forall\, a, b, c \in \mathbb{Z}$.

Uma relação que seja simultaneamente reflexiva, simétrica e transitiva é denominada relação de equivalência. Se R é uma **relação de equivalência** sobre um conjunto A, então R estabelece uma partição do conjunto A.

Suponha-se que R seja uma relação binária sobre A, e $A_i, i \geq 0$, uma partição de A induzida por R. Então, valem as seguintes propriedades:

- Se $(a, b) \in R$, então $a \in A_i, b \in A_j$ e $i = j$;
- Se $(a, b) \notin R$, então $a \in A_i, b \in A_j$ e $i \neq j$.

Exemplo 1.23 Considere-se o conjunto $\mathbb{Z}$ dos números inteiros e a relação binária:

$$\begin{aligned} Q &: \{(a, b) \in \mathbb{Z} \times \mathbb{Z} \mid a^2 = b^2\} \\ Q &= \{(0,0), (1,1), (1,-1), (-1,1), (-1,-1)...(n,n), (n,-n), (-n,n), (-n,-n)...\} \end{aligned}$$

É fácil verificar que Q é reflexiva, simétrica e transitiva. Logo, é uma relação de equivalência. Q induz à partição $\{A_0, A_1, ...\}$ de $\mathbb{Z}$, onde:

$$\begin{aligned} A_0 &= \{0, 0\} \\ A_1 &= \{1, -1\} \\ A_2 &= \{2, -2\} \\ &... \\ A_n &= \{n, -n\} \\ &... \end{aligned}$$

Quaisquer que sejam os números $a, b \in \mathbb{Z}$ considerados, se $(a, b) \in Q$, então a e b pertencem necessariamente ao mesmo conjunto A_i, para algum valor de $i \geq 0$. Se $(a, b) \notin Q$, a e b pertencerão sempre a conjuntos distintos desta partição de $\mathbb{Z}$. □

Diz-se que um conjunto é **fechado em relação a uma operação** se da aplicação dessa operação a quaisquer membros desse conjunto resultarem sempre elementos que também são membros do mesmo conjunto.

Exemplo 1.24 Considere-se o conjunto $X = \{x \in \mathbb{R} \mid x \geq 0\}$ e a operação unária $\sqrt{}$ (raiz quadrada). Qualquer que seja o elemento $x \in X$ considerado, $\sqrt{x}$ é sempre um elemento de X. Portanto, o conjunto X é fechado em relação à operação $\sqrt{}$.

Por outro lado, não se pode dizer o mesmo do conjunto $\mathbb{R}$, uma vez que a operação raiz quadrada não é definida para números negativos. Logo, o conjunto $\mathbb{R}$ não é fechado em relação à operação $\sqrt{}$. □

Exemplo 1.25 Considerem-se os conjuntos dos números inteiros $\mathbb{Z}$, dos números naturais $\mathbb{N}$ e as operações binárias de soma e subtração. Então, as seguintes afirmativas são verdadeiras:

- O conjunto $\mathbb{Z}$ é fechado em relação à operação de soma. De fato, da soma de quaisquer dois elementos de $\mathbb{Z}$ resulta sempre um elemento que também pertence ao conjunto $\mathbb{Z}$;
- O conjunto $\mathbb{Z}$ é fechado em relação à operação de subtração, pois da subtração de quaisquer dois elementos de $\mathbb{Z}$ resulta sempre um elemento que também pertence ao conjunto $\mathbb{Z}$;
- O conjunto $\mathbb{N}$ não é fechado em relação à operação de subtração: nem toda subtração de dois elementos arbitrários de $\mathbb{N}$ fornece como resultado um elemento que também pertença ao conjunto $\mathbb{N}$; Assim, por exemplo, se $1 \in \mathbb{N}$ e $2 \in \mathbb{N}$, $2 - 1 = 1 \in \mathbb{N}$, mas $1 - 2 = -1 \notin \mathbb{N}$;
- O conjunto $\mathbb{N}$ é fechado em relação à operação de soma.

□

1.3 Funções

Uma **função** é um mapeamento que associa elementos de um conjunto denominado **domínio** a elementos de um outro conjunto, chamado **co-domínio** ou **contradomínio**. Essa associação deve ser tal que cada elemento do domínio esteja associado a no máximo um elemento do conjunto co-domínio.

Formalmente, uma função entre um conjunto A (domínio) e um conjunto B (co-domínio) é definida como uma relação R entre esses conjuntos, de modo que:

$$\forall (a, b), (a, c) \in R, b = c$$

Portanto, o termo "função" refere-se um tipo particular de relação, em que cada elemento do domínio está associado a, no máximo, um único elemento do co-domínio. Em outras palavras, toda função é uma relação, mas nem toda relação é uma função.

Denota-se uma função f entre dois conjuntos X e Y por:

$$f : X \rightarrow Y$$

Exemplo 1.26 Considere-se f_1 e f_2 definidas abaixo:

$$\begin{aligned} f_1 &= \{(1,5),(2,3),(4,5),(8,1),(7,3)\} \\ f_2 &= \{(6,7),(9,0),(6,3),(4,3),(3,1)\} \end{aligned}$$

A relação f_1 é aderente à definição de função, ao passo que f_2 é uma relação mas não uma função, devido à presença simultânea dos pares $(6,7)$ e $(6,3)$, que associam o mesmo elemento 6 do domínio a dois elementos distintos do co-domínio (7 e 3). As Figuras 1.3 e 1.4 ilustram, respectivamente, as relações f_1 e f_2.

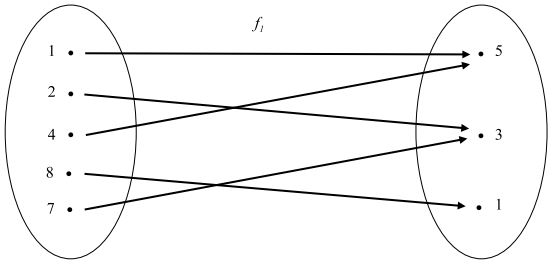

Figura 1.3: Relação que é também função

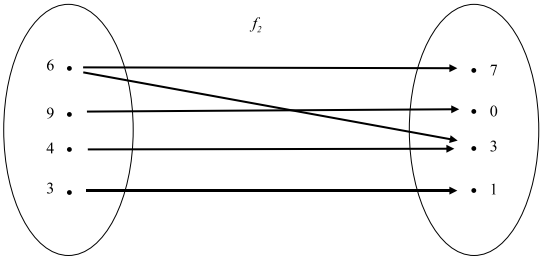

Figura 1.4: Relação que não é função

□

A associação estabelecida pela função f entre um elemento x do conjunto domínio X com um elemento y do conjunto co-domínio Y é denotada por:

$$f(x) = y$$

De maneira equivalente, diz-se que $(x, y) \in f$.

O **conjunto imagem** de f, denotado por I_f, é o conjunto formado por todos os elementos do co-domínio Y que estejam em correspondência com elementos de X, ou seja, $I_f \subseteq Y$. Formalmente,

$$I_f = \{y \in Y \mid y = f(x)\}$$

O elemento x é denominado **argumento** da função f, e y é denominado **imagem** de x pela **aplicação** de f. Funções com múltiplos argumentos são definidas como um mapeamento em que o conjunto domínio corresponde ao produto cartesiano de múltiplos conjuntos:

$$f : X_1 \times X_2 \times ... \times X_n \rightarrow Y$$

Funções com um, dois ou três argumentos são respectivamente denominadas funções unárias, binárias ou ternárias, e assim por diante.

Diz-se também que uma função que associa pares ordenados sobre um conjunto X, ou seja, elementos de X^2 com elementos do próprio conjunto X, é uma **função (operação) binária** sobre X.

Exemplo 1.27 Considere $f_1 : \mathbb{N} \rightarrow \mathbb{N}, f_1 = \{y \in \mathbb{N} \mid y = x^3, x \in \mathbb{N}\}$. A função f_1 é unária, pois associa cada elemento de $\mathbb{N}$ ao seu cubo. Portanto, $f_1 : \mathbb{N} \rightarrow \mathbb{N}$. Alguns dos infinitos elementos do conjunto definido por f_1 são: $(1, 1), (2, 8), (3, 27)$ etc. Denota-se $f_1(2) = 8$, ou ainda $(2, 8) \in f_1$. □

Exemplo 1.28 Seja $f_2 : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}, f_2 = \{z \in \mathbb{Z} \mid z = x + y;\ x, y \in \mathbb{Z}\}$. A função binária f_2 define a operação (função) de adição sobre o conjunto dos números inteiros $\mathbb{Z}$, sendo elementos de $f_2 : ((1,2),3), ((-3,7),4), ((0,5),5)$ etc. Escreve-se $f_2(-3,7) = 4$, ou ainda $((-3,7),4) \in f_2$. □

Uma função se diz uma **função total** (denotada pelo símbolo "$\rightarrow$") quando especifica associações para todos os elementos do conjunto domínio, sem exceção. Formalmente:

$$\forall\, x \in X, \exists\, y \in Y \mid y = f(x)$$

Exemplo 1.29 A Figura 1.5 ilustra o conceito de função total.

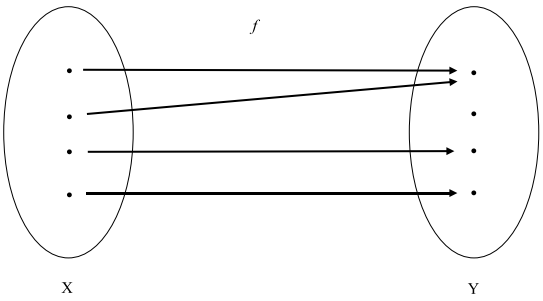

Figura 1.5: Função total

Notar que todos os elementos de X têm correspondência com algum elemento de Y. □

Exemplo 1.30 Sejam $X = \{0, 1, 2\}$ e $Y = \{a, b, c\}$, respectivamente, o conjunto domínio e o conjunto co-domínio da função $f_1 = \{(0, a), (1, b), (2, a)\}$. A função $f_1 : X \rightarrow Y$ é total, pois todos os elementos do conjunto domínio estão em correspondência com algum elemento do conjunto co-domínio. Neste caso, o conjunto imagem de f_1 é $\{a, b\}$. □

Quando uma função não é definida para todos os elementos de seu domínio, ela recebe a denominação de **função parcial** (denotada pelo símbolo "$\nrightarrow$"). Formalmente:

$$\exists\, x \in X \mid f(x) \text{ não é definida}$$

Exemplo 1.31 A Figura 1.6 ilustra o conceito de função parcial.

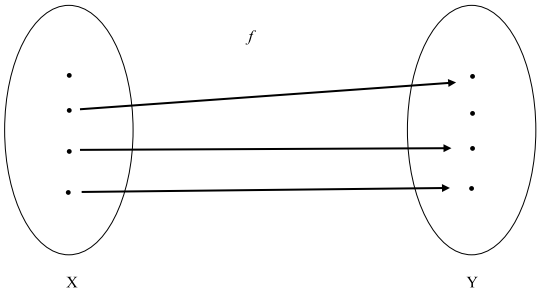

Figura 1.6: Função parcial

Notar a existência de um elemento de X sem correspondente em Y. □

Exemplo 1.32 Seja $X = \{0, 1, 2\}$, $Y = \{a, b, c\}$ e $f_2 = \{(0, b), (2, b)\}$. A função $f_2 : X \nrightarrow Y$ é parcial, pois não há associação do elemento "1" pertencente ao conjunto domínio a qualquer elemento do conjunto co-domínio. O conjunto imagem para essa função é $\{b\}$. □

Diz-se que uma função é **um-para-um**, ou simplesmente uma função **injetora**, quando elementos distintos do domínio X estiverem associados a elementos distintos do co-domínio Y, ou seja, quando não houver quaisquer dois elementos distintos do conjunto domínio associados ao mesmo elemento do conjunto imagem:

$$\forall\, x_1, x_2 \in X, x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$$

De maneira equivalente, uma função é dita **injetora** se cada elemento do conjunto co-domínio estiver associado a, no máximo, um elemento do conjunto domínio.

As Figuras 1.5 e 1.6 representam funções que são, respectivamente, não-injetora e injetora.

Exemplo 1.33 Seja $X = \{0, 1, 2\}$, $Y = \{a, b, c\}$ e $f_3 = \{(0, c), (1, b)\}$. A função $f_3 : X \rightarrow Y$ é injetora, pois não existe um mesmo elemento de Y associado a mais de um elemento de X. Por outro lado, a função f_2, definida no Exemplo 1.32, é parcial mas não injetora, pois o elemento b de seu conjunto imagem está simultaneamente associado aos elementos 0 e 2 do conjunto domínio. □

Uma função f é dita **sobrejetora** se todos os elementos do conjunto co-domínio estiverem associados a elementos do conjunto domínio, ou seja, se I_f, o conjunto imagem de f, for igual ao conjunto co-domínio de f:

$$\forall\, y \in Y, \exists\, x \in X \mid y = f(x)$$

Dito de outra forma, uma função é sobrejetora se todo elemento do conjunto co-domínio estiver associado a pelo menos um elemento do conjunto domínio.

Exemplo 1.34 As funções das Figuras 1.5 e 1.6 não são sobrejetoras. A Figura 1.7 ilustra uma função sobrejetora.

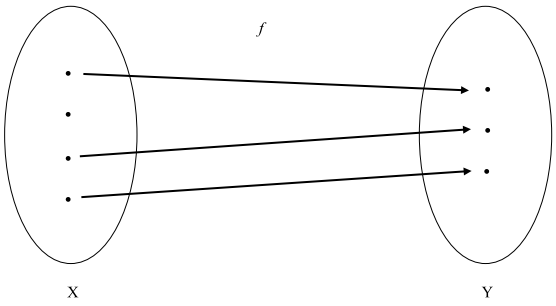

Figura 1.7: Função sobrejetora

Não há elemento de Y que não corresponda a algum elemento de X. □

Exemplo 1.35 Seja $X = \{0, 1, 2\}$, $Y = \{a, b, c\}$ e $f_4 = \{(0, c), (1, b), (2, a)\}$. A função $f_4 : X \rightarrow Y$ é sobrejetora, pois $Y = I_f = \{a, b, c\}$. Em adição, pode-se observar que f_4 é simultaneamente uma função total, injetora e sobrejetora, e também que as funções f_1 (Exemplo 1.30), f_2 (Exemplo 1.32) e f_3 (Exemplo 1.33) anteriormente definidas não são sobrejetoras. □

Uma função que seja simultaneamente total, injetora e sobrejetora recebe a denominação de função **bijetora**.

Exemplo 1.36 As funções das Figuras 1.5, 1.6 e 1.7 não são bijetoras. Em particular, a da Figura 1.5 é total, não-injetora e não-sobrejetora; a da Figura 1.6 é parcial, injetora e não-sobrejetora; e a da Figura 1.7 é parcial, injetora e sobrejetora. A Figura 1.8 ilustra uma função bijetora.

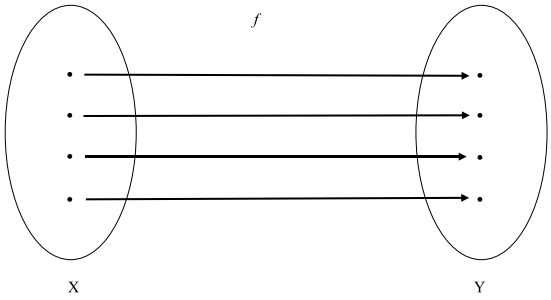

Figura 1.8: Função bijetora

Há uma correspondência biunívoca entre os elementos de X e os de Y. □

Exemplo 1.37 Seja $f_5 = \{(0, a), (1, b), (2, c)\}$. A função $f_5 : X \rightarrow Y$, assim como a função f_4 definida no Exemplo 1.35, é bijetora. As funções f_1 (Exemplo 1.30), f_2 (Exemplo 1.32) e f_3 (Exemplo 1.33) não são bijetoras. □

Exemplo 1.38 Considerem-se as funções adição, sobre o conjunto dos números naturais, divisão, sobre o conjunto dos números reais, e raiz quadrada, sobre o conjunto dos números inteiros:

- $+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$. Ela não é injetora, pois a soma de dois números naturais quaisquer pode corresponder à soma de outros números naturais distintos (por exemplo, ((3,4),7) e ((5,2),7)). É sobrejetora, pois todo número natural pode ser expresso como a soma de dois outros números naturais. É total, pois a cada par de números naturais sempre corresponde um outro número natural.

- $/ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$. Não é injetora, pois existem vários casos em que a divisão de dois números reais corresponde ao mesmo número real (por exemplo, os casos ((10.0,2.5),4.0) e ((20.0,5.0),4.0)). É sobrejetora, pois todos os números reais podem ser expressos como a divisão de dois outros números reais (por exemplo, todos os casos ((x,1.0),x)). Não é total, pois a divisão não é definida quando o denominador é zero (por exemplo, ((1,0),?)).

- $\sqrt{} : \mathbb{Z} \rightarrow \mathbb{Z}$. É injetora, pois não é possível que dois números inteiros diferentes tenham a mesma raiz inteira ((4,2), (9,3) e (3,?)). Não é sobrejetora, pois nem todo número inteiro corresponde à raiz quadrada de algum outro número inteiro (por exemplo, (?,-3). Não é total, pois a operação raiz quadrada não é definida para números inteiros negativos (por exemplo, (-2,?)).

A Tabela 1.1 resume estes resultados:

	Injetora?	Sobrejetora?	Total?
$+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$	Não	Sim	Sim
$/ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$	Não	Sim	Não
$\sqrt{} : \mathbb{Z} \rightarrow \mathbb{Z}$	Sim	Não	Não

Tabela 1.1: Propriedades das funções $+, /$ e $\sqrt{}$

□

1.4 Grafos

Um **grafo** é um par ordenado (V, A), em que V denota o conjunto de **vértices** (ou nós) do grafo e A denota uma relação binária sobre V, através da qual são especificados os **arcos** do grafo. Os arcos indicam associações entre os vértices do grafo. Dois vértices $v_i, v_j \in V$ tais que $(v_i, v_j) \in A$ são ditos vértices **adjacentes**.

A estrutura de um grafo pode ser melhor entendida com o auxílio de uma representação gráfica. Neste caso, os nós são denotados por círculos e os arcos por linhas que unem pares de vértices.

Exemplo 1.39 Sejam G_1, V_1 e A_1 conforme abaixo:

$$\begin{aligned} G_1 &= (V_1, A_1) \\ V_1 &= \{0, 1, 2, 3\} \\ A_1 &= \{(0,1), (0,2), (0,3), (1,3), (2,3)\} \end{aligned}$$

O grafo G_1 possui quatro vértices, respectivamente numerados de 0 a 3, e cinco arcos, que conectam pares de vértices, conforme especificado em A_1. Graficamente, G_1 pode ser ilustrado conforme a Figura 1.9.

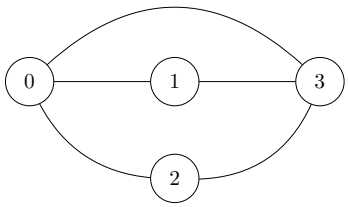

Figura 1.9: Grafo

□

Diz-se que um grafo é **orientado** quando os pares da relação binária A sobre V forem ordenados, ou seja, quando houver relação de ordem entre os elementos que formam os pares $(v_i, v_j) \in A$. Caso contrário, diz-se que o grafo é **não-orientado**.

Na prática, costuma-se convencionar que v_i **precede** v_j no par $(v_i, v_j) \in A$. Neste caso, v_i é denominado **predecessor** de v_j. Por outro lado, v_j **sucede** v_i neste mesmo par, e por isso é denominado **sucessor** de v_i. Diz-se também que o arco (v_i, v_j) **emerge** do vértice v_i (ou ainda "se inicia no", "parte do") e **atinge** o vértice v_j (ou "termina no", "chega ao", "alcança").

Grafos orientados empregam, em sua representação, setas associadas aos arcos, denotando, através do sentido dos arcos do grafo, a relação de ordem existente entre os nós unidos pelo arco em questão. A omissão das setas, em grafos não-orientados, equivale a considerar que todos os arcos representam conexões bidirecionais.

Exemplo 1.40 No Exemplo 1.39, o grafo G_1 está representado como um grafo não-orientado. Considerando-se desta vez o grafo G_1 como sendo um grafo orientado, e sem alterar qualquer aspecto de sua definição formal, o mesmo poderia ser representado graficamente conforme a Figura 1.10.

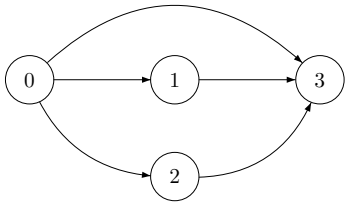

Figura 1.10: Grafo orientado G_1

□

Um grafo orientado é dito **ordenado** quando houver uma relação de ordem pré-convencionada sobre todos os arcos que emergem dos diversos vértices do grafo. Essa relação de ordem tem por objetivo estabelecer uma seqüência entre os diversos arcos que partem de um mesmo vértice, e não costuma ser definida explicitamente, uma vez que conjuntos não incorporam o conceito de seqüência.

Quando se deseja ordenar os arcos que emergem de cada vértice, é comum que se leve em conta como referência a seqüência em que os arcos comparecem na representação algébrica da função A. Eventualmente pode-se considerar uma seqüência diferente, desde que devidamente explicitada na representação do grafo.

Exemplo 1.41 Sejam:

$$\begin{aligned} G_2 &= (V_2, A_2) \\ V_2 &= \{a, b, c, d\} \\ A_2 &= \{(a,b),(b,a),(a,c),(a,d),(c,b),(d,c),(c,d)\} \end{aligned}$$

Suponha-se, para efeito didático, que A_2 fosse constituída de uma seqüência de pares ordenados (e não de um conjunto de pares ordenados), redigida de maneira análoga à representação do conjunto, porém sem as chaves:

$$(a,b),(b,a),(a,c),(a,d),(c,b),(d,c),(c,d)$$

Suponha-se, ainda, que um mesmo par ordenado não figure mais de uma vez na mesma seqüência, e que exista uma relação de ordem implícita entre os pares ordenados, de tal forma que $(a,b) < (b,a) < (a,c) < ... < (c,d)$. Isso facilita a percepção de uma relação de ordem definida para os arcos de G_2, conforme mostrado a seguir:

- Vértice a: inicialmente (a,b), depois (a,c) e por último (a,d)
- Vértice b: apenas (b,a)
- Vértice c: primeiro (c,b) depois (c,d)
- Vértice d: apenas (d,c)

Essa ordenação pode ser representada graficamente numerando-se os arcos do grafo, indicando-se assim a ordenação relativa dos arcos que partem de um mesmo vértice:

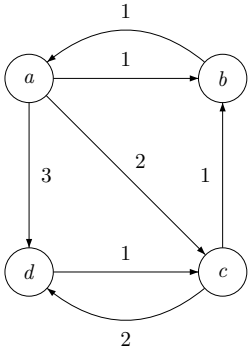

Figura 1.11: Grafo ordenado G_2

□

Três importantes conceitos estão relacionados a grafos orientados, sejam eles ordenados ou não. O número N_S de **ramificações de saída** (ou "*fan-out*") de um dado vértice de um grafo orientado indica a quantidade de arcos que partem do mesmo. De modo similar, o número N_E de **ramificações de entrada** (ou "*fan-in*") de um determinado vértice refere-se à quantidade de arcos do grafo que possuem o vértice em questão como destino. Vértices com $N_E = 0$ são denominados **vértices-base** ou **vértices-raiz**, e vértices com $N_S = 0$ são denominados **vértices-folha**.

Um **caminho** entre dois arcos, respectivamente denominados arcos **inicial** e **final**, em um grafo, é uma seqüência ordenada de arcos, de tal forma que o vértice predecessor de cada arco, à exceção do arco inicial, corresponde ao vértice sucessor do arco imediatamente anterior na seqüência ordenada.

O **comprimento** de um caminho é o número de arcos que o formam. Por definição, um caminho de comprimento zero é aquele que inicia e termina no mesmo vértice sem percorrer nenhum arco.

Um caminho é denominado **ciclo** se o vértice predecessor do primeiro arco coincidir com o vértice sucessor do último arco que o define. Grafos orientados que possuem pelo menos um ciclo são ditos grafos **cíclicos**. Caso contrário, são denominados grafos **acíclicos**.

Exemplo 1.42 Para o grafo G_2 da Figura 1.11 (Exemplo 1.41):

$$N_S(a) = 3, N_S(b) = 1, N_S(c) = 2, N_S(d) = 1$$
$$N_E(a) = 1, N_E(b) = 2, N_E(c) = 2, N_E(d) = 2$$

A seqüência $(a, c)(c, b)$ constitui um caminho, pois o vértice predecessor do segundo arco (c em (c, b)) é idêntico ao vértice sucessor do arco anterior (c em (a, c)), e seu comprimento é 2. A seqüência $(a, d)(c, d)(d, c)$ não constitui um caminho, pois o vértice predecessor do segundo arco ((c)) é diferente do vértice sucessor do arco anterior ((d)). O grafo G_2 é do tipo cíclico, pois é possível identificar inúmeros ciclos, dentre os quais $(a, b)(b, a)$ ou $(a, d)(d, c)(c, d)(d, c)(c, b)(b, a)$. □

Muitas vezes pode ser útil associar aos vértices de um grafo, aos arcos de um grafo, ou, eventualmente, a ambos, rótulos que representem informação adicional para a sua interpretação. Nesses casos, diz-se que o grafo é **rotulado**. Conforme o caso, caracteriza-se a rotulação de vértices ou então a rotulação de arcos do grafo.

Uma **rotulação de vértices** é definida como sendo uma função f_V que associa os elementos de V a elementos de um conjunto R_V, denominado alfabeto de rotulação de vértices. De modo análogo, uma **rotulação de arcos** é realizada através de uma função f_A que associa elementos de A a elementos de um conjunto R_A, denominado alfabeto de rotulação de arcos.

Exemplo 1.43 Sejam:

$$\begin{aligned} G_3 &= (V_3, A_3) \\ V_3 &= \{0, 1, 2\} \\ A_3 &= \{(0,1), (1,2), (0,2)\} \end{aligned}$$

Uma possível rotulação simultânea de vértices e arcos em G_3 seria:

- $f_V = \{(0, \phi), (1, \gamma), (2, \psi)\}$, com $R_V = \{\phi, \gamma, \psi\}$
- $f_A = \{((0,1), \Phi), ((1,2), \Gamma), ((0,2), \Psi)\}$, com $R_A = \{\Phi, \Gamma, \Psi\}$

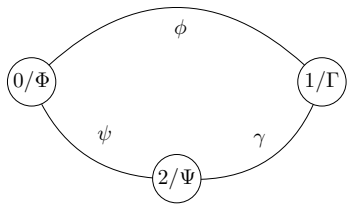

Figura 1.12: Grafo rotulado G_3

Esquematicamente, essa rotulação pode ser representada conforme mostra a Figura 1.12. □

Quanto à sua natureza, os grafos podem ser classificados em grafos orientados ou grafos não-orientados, e também em grafos rotulados ou grafos não-rotulados. Os grafos orientados podem ainda ser classificados em grafos ordenados ou grafos não-ordenados, e ainda como grafos cíclicos ou grafos acíclicos:

$$\text{Grafos} \begin{cases} \text{Orientados / Não-orientados} \begin{cases} \text{Ordenados / Não-ordenados} \\ \text{Cíclicos / Acíclicos} \end{cases} \\ \\ \text{Rotulados / Não-rotulados} \end{cases}$$

1.5 Árvores

São especialmente importantes, no estudo das linguagens formais, e muito aplicados na prática, para a análise e construção de compiladores, os grafos acíclicos orientados e as árvores. Estas, por sua vez, constituem um caso particular dos grafos acíclicos orientados ordenados.

Uma árvore ordenada, ou simplesmente uma **árvore**, é um grafo acíclico orientado e ordenado que possui as seguintes características adicionais:

- Há apenas um vértice r tal que $N_E(r) = 0$. Este vértice diferenciado é denominado **raiz** da árvore.

- Todos os demais vértices possuem $N_E = 1$.
- Para cada vértice há sempre um único caminho que o liga à raiz da árvore.

Para vértices a e b que fazem parte de um mesmo caminho em uma árvore, diz-se que a é **ancestral** de b se for possível atingir b a partir de a. Nesse caso, b é dito **descendente** de a. Quando entre a e b não houver nenhum vértice intermediário, diz-se que a e b são adjacentes. Nessa situação, diz-se ainda que o vértice a é ancestral direto, ou **pai**, do vértice b, e que este é descendente direto, ou **filho**, do vértice a. O ancestral mínimo comum de dois vértices a e b corresponde ao (único) antecessor de ambos que seja também descendente de todos os antecessores comuns de a e b.

Vértices tais que $N_S = 0$ são denominados **folhas** da árvore. Os demais são denominados **vértices internos**. Inclui-se, por essa definição, entre os vértices internos de uma árvore, o vértice-raiz dessa árvore.

A **profundidade** de um vértice em uma árvore é o comprimento do caminho iniciado em sua raiz, e que termina no referido vértice. A profundidade de uma árvore é definida como sendo a maior dentre as profundidades de seus vértices.

Árvores costumam ser representadas esquematicamente com a raiz na parte superior da figura, e com os arcos e demais vértices "crescendo" para baixo. Normalmente, as representações esquemáticas das árvores não incorporam indicações sobre a orientação e a ordenação dos arcos, as quais neste caso se tornam implícitas, podendo ser inferidas a partir da própria figura, de acordo com as seguintes convenções, usualmente adotadas: todos os arcos "apontam" para baixo e são implicitamente ordenados da esquerda para a direita.

Exemplo 1.44 Considere-se a árvore da Figura 1.13:

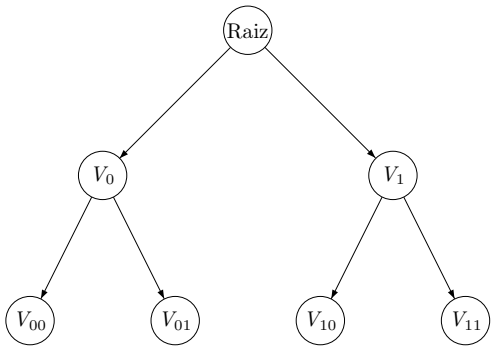

Figura 1.13: Árvore para o Exemplo 1.44

Neste exemplo, o vértice V_1 é ancestral direto (pai) de V_{11}. Este, por sua vez é descendente direto (filho) de V_1. O vértice "Raiz" é ancestral de V_{00} e o vértice V_{01} é descendente de "Raiz". "Raiz" é também o mínimo ancestral comum dos vértices V_{00} e V_{11}, que por sua vez são folhas dessa árvore e apresentam profundidade igual a 2. V_0 e V_1 são vértices internos, e possuem profundidade 1. A profundidade desta árvore é igual a 2. □

1.6 Teoremas e Demonstrações

Linguagens formais e autômatos constituem sistemas matemáticos formais, nos quais inúmeras propriedades, em geral formuladas como **teoremas**, podem ser inferidas a partir de verdades previamente conhecidas ou admitidas por hipótese, por intermédio de raciocínios lógicos expressos como **demonstrações**. Tais propriedades sintetizam grande parte dos resultados mais importantes da teoria na área, e são fundamentais para o aprendizado e a aplicação do conhecimento adquirido.

A demonstração de teoremas sobre um dado conjunto geralmente exige a prova formal de que uma certa propriedade é satisfeita por todos os membros desse conjunto. Quando se trata de conjuntos com uma quantidade reduzida de elementos, é possível realizar demonstrações particulares para cada elemento individual deste conjunto (que às vezes se reduzem a simples verificações), garantindo assim que a propriedade seja válida para todos os elementos do conjunto.

Há, no entanto, uma óbvia dificuldade prática na aplicação dessa técnica para conjuntos finitos com cardinalidades elevadas. Além disso, esse método exaustivo de demonstração evidentemente não se aplica a conjuntos infinitos.

Para tais casos, devem-se buscar formas alternativas que permitam verificar, segundo critérios de economia e de factibilidade, a validade de proposições efetuadas acerca do sistema formal em estudo. Dentre as técnicas mais largamente empregadas para tal finalidade destacam-se as demonstrações por indução matemática e as provas por contradição, estas também conhecidas por demonstrações por redução ao absurdo.

Indução Matemática

O princípio da **indução matemática** foi estabelecido com o intuito de permitir a generalização de uma propriedade P para um conjunto infinito de elementos X. Informalmente, indução é definida como uma "operação mental que consiste em se estabelecer uma verdade universal ou proposição geral com base no conhecimento de certo número de dados singulares ou de proposições de menor generalidade" (Novo Dicionário Aurélio da Língua Portuguesa).

Formalmente, o princípio da indução é estabelecido da seguinte forma:

- Inicialmente, elege-se um elemento ou subconjunto de elementos destacados de X, denominado **base da indução**, e demonstra-se que a propriedade P é válida neste caso particular. Geralmente, essa parte da prova é trivial.

- Admite-se, em seguida, que a propriedade seja válida para subconjuntos finitos de X que contenham o elemento utilizado para demonstrar a base da indução. Essa etapa é conhecida como **hipótese indutiva**, e é formulada, de maneira recorrente, nos próprios termos da propriedade que se deseja demonstrar.

- A seguir, confinado no fato de que, de maneira recorrente, a hipótese indutiva é verdadeira, demonstra-se que, se P é válida para um determinado elemento ou subconjunto de X, então P continuará sendo válida quando se acrescenta mais um elemento ao conjunto X (ou seja, P é válida para subconjuntos sucessivamente mais abrangentes de X). Essa etapa recebe a denominação de **passo indutivo**, e realiza a generalização da propriedade proposta.

Exemplo 1.45 Deseja-se provar que a propriedade P_0 abaixo é válida para todos os números naturais maiores que 1:

$P_0(n) : 1 + 2^n < 3^n, \forall\, n \geq 2.$

Base da indução:

$n = 2.$

$P_0(2) = 1 + 2^2 = 5 < 9$. Portanto, P_0 é válida para $n = 2$.

Hipótese indutiva:

$P_0(k)$ é válida para $k \geq 2$, ou seja, $1 + 2^k < 3^k, k \geq 2$.

Passo indutivo:

$P_0(k)$ implica $P_0(k+1), k \geq 2$.

Prova:

Pela hipótese indutiva: $P_0(k) = 1 + 2^k < 3^k, k \geq 2$.

Multiplicando-se ambos os membros da desigualdade por $2 : 2 + 2^{k+1} < 2 * 3^k$.

Subtraindo-se 1 de ambos os membros da desigualdade: $1 + 2^{k+1} < 2 * 3^k - 1$.

Como $2x - 1 < 3x, \forall\, x \geq 1$ (porque $3x = (2+1)x = 2x + 2$, logo $2x - 1 < 2x + 2$ e $-1 < 2$ é verdadeiro), então $1 + 2^{k+1} < 2 * 3^k - 1 < 3 * 3^k = 3^{k+1}$.

Portanto, $1 + 2^{k+1} < 3^{k+1} = P_0(k+1)$, ou seja, $P_0(n), \forall\, n \geq 2$.

A título de complementação deste exemplo, pode-se demonstrar, também por indução, que a propriedade $2n - 1 < 3n, n \geq 1$, empregada na demonstração acima, é verificada no intervalo especificado. Na prática, a necessidade de se demonstrar passos ou hipóteses intermediárias em geral varia, conforme seja ou não intuitivo aceitar as afirmações apresentadas.

$P_1(n) = 2n - 1 < 3n, \forall\, n \geq 1.$

Base da indução:

$n = 1.$

$P_1(1) = 2 * 1 - 1 = 1 < 3.$

Hipótese indutiva:

$P_1(k)$ é válida para $k \geq 1$, ou seja, $2k - 1 < 3k, \forall\, k \geq 1$.

Passo indutivo:

$P_1(k)$ implica $P_1(k+1), k \geq 1$.

Prova:

Pela hipótese indutiva: $P_1(k) = 2k - 1 < 3k, k \geq 1$.

Somando-se 2 a ambos os membros da desigualdade: $2k + 1 < 3k + 2$.

Como: $3x < 3x + 1, \forall\, x \geq 1$

Então: $2k + 1 < 3k + 2 < 3k + 3$, ou seja, $2k + 1 < 3k + 3$

Logo: $2(k+1) - 1 < 3(k+1) = P_1(k+1)$, ou $P_1(n), \forall\, n \geq 1$. □

Exemplo 1.46 Demonstrar, por indução, que $P_2(n) : \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$.

Base da indução:

$n = 0.$

$$P_2(0) : \sum_{i=0}^{0} i = \frac{0(0+1)}{2} = 0$$

Hipótese indutiva:

$$P_2(k) : \sum_{i=0}^{k} i = \frac{k(k+1)}{2}, k \geq 0$$

Passo indutivo:

$P_2(k)$ implica $P_2(k+1), k \geq 0$.

Prova:

Pela hipótese indutiva: $\sum_{i=0}^{k} i = \frac{k(k+1)}{2}, k \geq 0$

Somando-se (k+1) a ambos os membros da igualdade:

$$\sum_{i=0}^{k} i + (k+1) = \sum_{i=0}^{k+1} i = \frac{k(k+1)}{2} + (k+1)$$

Desmembrando: $\frac{k(k+1)}{2} + (k+1) = \frac{k^2 + k + 2k + 2}{2} = \frac{k^2 + 3k + 2}{2}$

Fatorando: $\frac{k^2 + 3k + 2}{2} = \frac{(k+1)(k+2)}{2} = \frac{(k+1)((k+1)+1)}{2}$

Logo: $\sum_{i=0}^{k+1} i = \frac{(k+1)((k+1)+1)}{2} = P_2(k+1)$ □

Exemplo 1.47 Demonstrar, por indução, que $P_3(n) = \sum_{i=0}^{n} i^3 = \left(\sum_{i=0}^{n} i\right)^2, \forall\, n \geq 0$

Base da indução:

$n = 0$.

$\sum_{i=0}^{0} i^3 = \left(\sum_{i=0}^{0} i\right)^2 = 0$

Hipótese indutiva:

$\sum_{i=0}^{k} i^3 = \left(\sum_{i=0}^{k} i\right)^2, \forall\, k \geq 0$

Passo indutivo:

$P_3(k)$ implica $P_3(k+1), k \geq 0$.

Prova:

Pela hipótese indutiva: $\sum_{i=0}^{k} i^3 = \left(\sum_{i=0}^{k} i\right)^2, \forall\, k \geq 0$

Somando-se $(k+1)^3$ a ambos os membros da igualdade:

$\sum_{i=0}^{k} i^3 + (k+1)^3 = \left(\sum_{i=0}^{k} i\right)^2 + (k+1)^3 = \sum_{i=0}^{k+1} i^3$

De acordo com o exemplo anterior: $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}, \forall\, n \geq 0$

Logo: $\left(\sum_{i=0}^{k} i\right)^2 + (k+1)^3 = \left(\frac{k(k+1)}{2}\right)^2 + (k+1)^3$

Desmembrando: $\left(\frac{k(k+1)}{2}\right)^2 + (k+1)^3 = \frac{k^2(k+1)^2}{4} + (k+1)(k+1)^2$

Fatorando: $\frac{k^2(k+1)^2}{4} + (k+1)(k+1)^2 = (k+1)^2 \frac{k^2+4k+4}{4}$

Fatorando novamente:

$(k+1)^2 \frac{k^2+4k+4}{4} = (k+1)^2 \left(\frac{k+2}{2}\right)^2 = \left((k+1)\left(\frac{k+2}{2}\right)\right)^2$

Como: $\left((k+1)\left(\frac{k+2}{2}\right)\right)^2 = \left((k+1)\left(\frac{(k+1)+1}{2}\right)\right)^2 = \left(\sum_{i=0}^{k+1} i\right)^2$

Então: $\sum_{i=0}^{k} i^3 + (k+1)^3 = \sum_{i=0}^{k+1} i^3 = \left(\sum_{i=0}^{k+1} i\right)^2 = P_3(k+1)$ □

Os Exemplos 1.45, 1.46 e 1.47 demonstram a validade de proposições sobre subconjuntos dos números naturais. Na prática, no entanto, proposições sobre outros tipos de conjuntos, e não apenas $\mathbb{N}$, podem ser formuladas, mantendo-se a aplicabilidade integral dos princípios e das técnicas ilustradas nos exemplos.

Contradição

Uma outra técnica bastante popular utilizada na demonstração de teoremas é a demonstração por contradição.

A essência da técnica da demonstração por **contradição** (ou **redução ao absurdo**) consiste em adotar como base da demonstração a negação da hipótese formulada e, através de manipulações lógicas, mostrar que a negação dessa hipótese conduz a um paradoxo, muitas vezes correspondente à sua própria contradição. Dessa forma, a hipótese efetuada não pode ser considerada falsa, devendo, portanto, ser considerada verdadeira.

Exemplo 1.48 Deseja-se provar, por redução ao absurdo, que $\sqrt{2}$ não pode ser expressa como um número racional, ou seja, como fração cujo numerador e denominador, respectivamente p e q, são números inteiros (conforme [54]). Admitindo-se a hipótese como verdadeira (negação da hipótese original), então:

$\sqrt{2} = \frac{p}{q}$

Pode-se considerar, sem perda de generalidade, que p e q não possuem fatores comuns. Caso haja fatores comuns, simplifica-se a fração antes de iniciar o procedimento. Elevando-se ambos os lados da igualdade ao quadrado, obtém-se:

$2 = \frac{p^2}{q^2}$, ou $p^2 = 2q^2$

Portanto, p^2 e, conseqüentemente, p são números pares. Substituindo-se p por $2m$ na equação acima, temos:

$(2m)^2 = 2q^2$, ou $q^2 = 2m^2$

Isso mostra que q^2 e, portanto, q também são números pares. Ora, se p e q são pares, isso contradiz a hipótese original de que entre eles não haveria fatores comuns. Dessa forma, a hipótese não pode ser verdadeira, concluindo-se que a raiz quadrada de 2 não pode ser um número racional□

1.7 Conjuntos Enumeráveis

Quando se estudam os conjuntos, freqüentemente torna-se necessário compará-los entre si em relação à quantidade de elementos neles contidos, ou seja, à sua **cardinalidade**.

A cardinalidade de um conjunto é uma medida da quantidade de elementos contidos no mesmo, ou seja, da grandeza que intuitivamente é conhecida como "tamanho" do conjunto.

Trata-se de um conceito de fácil compreensão quando referente a conjuntos finitos. Nesse caso, diz-se que dois conjuntos A e B têm a mesma cardinalidade se eles possuírem a mesma quantidade de elementos, ou seja, $\big|A\big| = \big|B\big|$. Se A possuir mais elementos que B, escreve-se $\big|A\big| > \big|B\big|$.

A cardinalidade de um conjunto finito é, portanto, simplesmente o número natural que informa a quantidade de elementos que compõem esse conjunto. Quando se trata de conjuntos finitos, tais resultados são intuitivos e, até certo ponto, óbvios. Por exemplo, se X for um subconjunto próprio de Y, então ter-se-á sempre $\big|X\big| < \big|Y\big|$.

Exemplo 1.49 Considerem-se os conjuntos finitos $A = \{a, b, c, d\}$ e $B = \{0, 1, 2, 3, 4, 5\}$. Então, $\big|A\big| = 4, \big|B\big| = 6$ e $\big|A\big| < \big|B\big|$. □

De que forma seria, por outro lado, possível comparar o "tamanho" de dois conjuntos infinitos? Assim como no caso dos conjuntos finitos, dois conjuntos infinitos também podem possuir a mesma cardinalidade, bastando para isso que seja possível identificar uma correspondência biunívoca entre os elementos de ambos os conjuntos.

Formalmente, diz-se que dois conjuntos A e B quaisquer, finitos ou infinitos, possuem a mesma cardinalidade, ou seja, $\big|A\big| = \big|B\big|$, se for possível definir entre eles uma função bijetora.

Exemplo 1.50 Sejam $A = \{a, b, c\}$ e $B = \{7, 3, 6\}$. Neste exemplo, A e B possuem a mesma cardinalidade, pois $\big|A\big| = \big|B\big| = 3$. Note-se que é possível definir uma função bijetora de A para B: $\{(a, 7), (b, 3), (c, 6)\}$. Naturalmente, muitas outras funções bijetoras também podem ser definidas entre esses dois conjuntos. □

Exemplo 1.51 Sejam $A = \{a \mid a \text{ é ímpar}, 1 \leq a \leq 100\}$ e $B = \{b \mid b \text{ é par}, 1 \leq b \leq 100\}$. A e B são conjuntos finitos que possuem a mesma cardinalidade, pois a função $f(a) = a + 1$ é bijetora, mapeando os elementos do conjunto A nos elementos do conjunto B. Neste caso, $\big|A\big| = \big|B\big| = 50$□

Exemplo 1.52 Considere-se o conjunto dos números inteiros $\mathbb{Z}$ e o subconjunto de $\mathbb{Z}$ composto apenas pelos números ímpares. Trata-se, naturalmente, de dois conjuntos infinitos, sendo o segundo um subconjunto próprio do primeiro. Porém, de acordo com a definição, embora isso pareça paradoxal, os dois conjuntos possuem a mesma cardinalidade, já que a função bijetora $2 * i + 1$, onde $i \in \mathbb{Z}$, mapeia univocamente cada elemento de $\mathbb{Z}$ em um único elemento do conjunto dos números ímpares□

Do Exemplo 1.52 pode-se observar facilmente que, diferentemente do que ocorre com conjuntos finitos, é possível, para conjuntos infinitos, definir subconjuntos próprios com a mesma cardinalidade do conjunto original.

Caso não seja possível identificar pelo menos uma função bijetora entre dois conjuntos A e B quaisquer, é ainda possível que se constate a existência de uma função total e injetora de A para B. Neste caso, diz-se que $\big|A\ \big| \leq \big|B\big|$. Se, além disso, for possível provar a inexistência de uma função bijetora de A para B, então $\big|A\big| < \big|B\big|$.

Diz-se que um conjunto é **enumerável**, ou simplesmente **contável**, se ele possuir um número finito de elementos, ou então, no caso de ser infinito, se ele possuir a mesma cardinalidade que o conjunto dos números naturais $\mathbb{N}$. Conjuntos infinitos X tais que $\big|X\big| \neq \big|\mathbb{N}\big|$ são ditos **não-enumeráveis** ou **não-contáveis**.

O conceito de conjuntos enumeráveis está diretamente relacionado ao conceito intuitivo de "seqüencialização" dos elementos de um conjunto, com o objetivo de permitir a sua contagem. Na prática, a operação de contagem de elementos de um conjunto pode ser definida como o estabelecimento de uma correspondência única (função bijetora) entre o conjunto dos números naturais e o conjunto cujos elementos se pretenda contar.

A seqüencialização é uma operação que visa estabelecer uma relação de ordem entre os elementos de um conjunto (efetuar a sua ordenação) para permitir a associação unívoca de cada um de seus elementos com os correspondentes elementos de $\mathbb{N}$.

Exemplo 1.53 O conjunto dos números inteiros $\mathbb{Z}$ é um exemplo de conjunto infinito enumerável. A ordenação apresentada na Tabela 1.2 ilustra uma seqüencialização que permite associar os elementos de $\mathbb{Z}$ com os de $\mathbb{N}$:

$\mathbb{Z}$	0	1	-1	2	-2	3	-3	...
$\mathbb{N}$	0	1	2	3	4	5	6	...

Tabela 1.2: Bijeção entre $\mathbb{N}$ e $\mathbb{Z}$

Essa associação também pode ser representada por meio da função:

$$f(n) = (-1)^{n+1} * \frac{n + (n \bmod 2)}{2}$$

□

Exemplo 1.54 O conjunto formado pelos pares ordenados $(x, y) \in \mathbb{N} \times \mathbb{N}$, com $x > y$, também constitui um exemplo de conjunto infinito enumerável. Isso pode ser percebido com o auxílio da Tabela 1.3, em que um arranjo bidimensional permite visualizar a seqüencialização desses pares, de modo que seja possível estabelecer a sua associação com os elementos de $\mathbb{N}$.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	...
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	...
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	...
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	...
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	...
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	...
⋮	⋮	⋮	⋮	⋮	⋮	

Tabela 1.3: $\{(x, y) \in \mathbb{N} \times \mathbb{N} \mid x > y\}$ é um conjunto enumerável

A associação com $\mathbb{N}$ pode ser feita imaginando-se uma linha que percorra todos os elementos desta matriz a partir do canto superior esquerdo, conforme a seqüência geométrica mostrada na Tabela 1.3. Desse modo, a seguinte seqüência de pares é obtida:

$$(1,0),(2,0),(3,0),(2,1),(3,1),(4,0),(5,0),(4,1),(3,2)...$$

Tal seqüência pode ser facilmente colocada em correspondência com os elementos de $\mathbb{N}$, conforme ilustrado na Tabela 1.4.

$\mathbb{Z} \times \mathbb{Z}$	(1,0)	(2,0)	(3,0)	(2,1)	...
$\mathbb{N}$	0	1	2	3	...

Tabela 1.4: Bijeção entre $\mathbb{N}$ e $\mathbb{N} \times \mathbb{N}$

Técnica semelhante pode ser usada para demonstrar que o conjunto $\mathbb{N} \times \mathbb{N}$ e o conjunto dos números racionais também são enumeráveis. Neste último caso, em particular, basta considerar o elemento $(x, y) \in \mathbb{N} \times \mathbb{N}$ como uma representação da fração x/y (a fim de evitar o denominador zero, a primeira coluna do arranjo deve ser omitida). □

Exemplo 1.55 O conjunto $\mathbb{R}$, composto pelos números reais, constitui um exemplo de conjunto infinito não-enumerável, uma vez que, como demonstrado a seguir, $\left|\mathbb{R}\right| \neq \left|\mathbb{N}\right|$. Para efetuar essa demonstração, será considerado o seguinte subconjunto de $\mathbb{R}$:

$$S = \{x \in \mathbb{R} \mid 0 < x < 1\}$$

A prova de que $\mathbb{R}$ é não-enumerável é efetuada em dois passos: inicialmente demonstra-se que S possui a mesma cardinalidade que $\mathbb{R}$, e a seguir demonstra-se que S é um conjunto não-enumerável. O fato de que $\left|S\right| = \left|R\right|$ pode ser constatado pela existência da função bijetora f, apresentada a seguir, a qual permite associar univocamente elementos de S com elementos de $\mathbb{R}$:

$$f(x) = \begin{cases} \dfrac{1}{2x} - 1, & 0 < x < 0,5 \\ \dfrac{1}{2(x-1)} + 1, & 0,5 \leq x < 1 \end{cases}$$

A prova de que S é um conjunto não-enumerável é feita por contradição, ou seja, mostrando-se que, qualquer que seja a seqüencialização proposta para os elementos de S, sempre será possível identificar um novo elemento de S que não pertence à seqüência apresentada. Desse modo, a hipótese original de que S é um conjunto enumerável deve ser considerada inválida.

Admita-se que exista uma seqüencialização de S de tal modo que seja possível associar cada elemento desse conjunto univocamente a elementos de $\mathbb{N}$. Assim, seria obtida uma associação do tipo ilustrado pela Tabela 1.5.

$\mathbb{R}$	$\mathbb{R}_0$	$\mathbb{R}_1$	$\mathbb{R}_2$	$\mathbb{R}_3$	...
$\mathbb{N}$	0	1	2	3	...

Tabela 1.5: Bijeção hipotética entre $\mathbb{N}$ e $\mathbb{R}$

Suponha-se, sem perda de generalidade, que todos os elementos de S sejam denotados através de representações com um número infinito de casas de decimais significativas. Assim, por exemplo, em vez de escrever $0,1$, adota-se a representação equivalente $0,09999999999999999999999...$

Se cada elemento $R_i \in S$ pode ser escrito como sendo uma seqüência infinita do tipo: $0, d_{i_0} d_{i_1} d_{i_2} d_{i_3} ... d_{i_n} ...$, então a construção do novo elemento:

$$0, x_0 x_1 x_2 x_3 ... x_n ..., x_j \neq d_{jj}, x_j \neq 0$$

é suficiente para provar que o mesmo não pertence à seqüência originalmente proposta, uma vez que esse novo elemento difere em pelo menos uma casa decimal de cada um dos elementos inicialmente considerados, sendo, portanto, diferente de todos eles. Logo, S não é enumerável e, conseqüentemente, $\mathbb{R}$ também não. □

Como está mostrado no Exemplo 1.55, nem todos os conjuntos infinitos possuem a mesma cardinalidade. Assim, apesar de $\mathbb{N}$ e $\mathbb{R}$ possuírem uma quantidade infinita de elementos, é intuitivo que $\mathbb{R}$ possui uma quantidade muito maior de elementos que $\mathbb{N}$, ou seja, $\left|\mathbb{R}\right| > \left|\mathbb{N}\right|$, impedindo que seja estabelecida uma função bijetora entre ambos.

Esses são alguns dos resultados da Teoria dos Números Transfinitos ([57]), desenvolvida no final do século XIX pelo matemático russo Georg Cantor (1845-1918), de acordo com a qual os **números transfinitos** representam quantidades não-finitas ordenadas de forma crescente. Tais quantidades são representadas por $\aleph_0, \aleph_1 ... \aleph_n ...$ [1], de tal forma que $\aleph_{i-1} < \aleph_i < \aleph_{i+1}$, para $i \geq 1$. Além disso, $\aleph_0 = \left|\mathbb{N}\right|$ e $\aleph_1 = \left|\mathbb{R}\right|$.

Outros exemplos de conjuntos infinitos enumeráveis são o conjunto dos números racionais e o conjunto de todas as cadeias que podem ser formadas pela concatenação de símbolos de um conjunto finito Σ. Já o conjunto formado por todos os subconjuntos de $\mathbb{N}$, ou seja, o conjunto $2^{\mathbb{N}}$, é não-enumerável.

Formalmente, um conjunto X é dito **infinito** se for possível identificar um subconjunto próprio de X, por exemplo, Y, tal que $\left|X\right| = \left|Y\right|$.

Exemplo 1.56 No Exemplo 1.55, o fato de que $S \subset \mathbb{R}$ e $\left|S\right| = \left|\mathbb{R}\right|$ é suficiente para garantir que $\mathbb{R}$ é um conjunto infinito. □

Exemplo 1.57 Considere-se o conjunto dos números naturais $\mathbb{N}$. Deseja-se demonstrar que $\mathbb{N}$ é infinito com o auxílio do subconjunto próprio $\mathbb{N} - \{0\}$. Não é difícil perceber que esses dois conjuntos possuem a mesma cardinalidade, uma vez que a função $n + 1, n \in \mathbb{N}$ mapeia univocamente cada elemento de $\mathbb{N}$ em elementos do subconjunto próprio $\mathbb{N} - \{0\} : 0 \rightarrow 1, 1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 6, 6 \rightarrow 7...$

Assim, apesar de $\mathbb{N} - \{0\}$ possuir um elemento a menos que $\mathbb{N}$, na verdade ambos possuem a mesma cardinalidade, o que confirma $\mathbb{N}$ como conjunto infinito. □

A seguir serão apresentados e demonstrados alguns dos principais resultados teóricos sobre conjuntos enumeráveis e não-enumeráveis.

Teorema 1.3 (Teorema de Cantor) *Seja A um conjunto qualquer, $\left|A\right| = \aleph_i$. Então $\left|2^A\right| > \left|A\right|$ e $\left|2^A\right| = \aleph_{i+1}$.*

Justificativa Constata-se com facilidade a existência de pelo menos uma função f, que associa cada elemento $x \in A$ com um elemento $f(x) \in 2^A$, e que seja injetora e total. Logo, é possível concluir que $\left|A\right| \leq \left|2^A\right|$. Para provar que $\left|A\right| < \left|2^A\right|$, é suficiente mostrar que não existe função bijetora de A para 2^A.

Suponha-se que exista tal bijeção. Nesse caso, pode-se afirmar que todo e qualquer elemento $x \in A$ está associado a um elemento distinto $f(x) \in 2^A$. Considere-se agora o seguinte subconjunto de A:

$$S = \{x \in A \mid x \notin f(x)\}$$

[1] ℵ (aleph) é a primeira letra do alfabeto hebraico.

De acordo com a hipótese formulada (de que existe uma bijeção entre os conjuntos), é esperado que $S = f(x_i)$ para algum $x_i \in A$. Tal conclusão, se verdadeira, acarretaria as seguintes conseqüências, de forma exclusiva:

- Se $x_i \in S$, e como $S = f(x_i)$, por hipótese, então $x_i \notin S$, o que constitui uma contradição;
- Se $x_i \notin S$, e como $S = \{x \in A \mid x \notin f(x)\}$, por definição, então $x_i \in S$, o que também é uma contradição.

Qualquer que seja o caso, resulta uma contradição. Logo, a hipótese inicialmente formulada é falsa e disso conclui-se não existir qualquer bijeção entre A e 2^A. Portanto, $|A| < |2^A|$. ■

O Teorema 1.3 demonstra que conjuntos infinitos de cardinalidades sucessivamente maiores podem ser obtidos pela aplicação sucessiva da operação conjunto-potência. Considere os conjuntos $A, B = 2^A, C = 2^B, D = 2^C$ etc. Então, $|A| < |B| < |C| < |D| < ...$

De acordo com a teoria de Cantor, $\mathbb{N}$ é o conjunto que possui a menor cardinalidade entre todos os conjuntos infinitos, a qual é denotada por $\aleph_0$, o primeiro número da sua série transfinita. Por conseqüência, $|\mathbb{N}| < |2^{\mathbb{N}}|$. Por outro lado, conforme foi visto anteriormente, $|\mathbb{N}| < |\mathbb{R}|$, o que sugere a questão: "será que $|\mathbb{R}| = |2^{\mathbb{N}}|$?".

De fato, este resultado pode ser provado como sendo verdadeiro. Além disso, $|\mathbb{R}|$ e $|2^{\mathbb{N}}|$ correspondem ao segundo número transfinito conhecido, na seqüência de Cantor, o qual é denotado por $\aleph_1$. Por outro lado, não se sabe da existência de algum conjunto X, tal que $\aleph_0 < |X| < \aleph_1$.

Teorema 1.4 ($|B|, B \subseteq A, |A| = \aleph_0$) *Sejam A e B dois conjuntos, $B \subseteq A$. Se $|A| = \aleph_0$, então $|B| \leq \aleph_0$.*

Justificativa Se $|A| = \aleph_0$, então existe uma função bijetora entre o conjunto dos números naturais $\mathbb{N}$ e o conjunto A (e vice-versa). Logo, existe uma função injetora e total f_1 que associa elementos de A e $\mathbb{N}$, conforme a Tabela 1.6.

A:	a_0	a_1	a_2	...	a_n	...
f_1 :	↓	↓	↓		↓	
$\mathbb{N}$	0	1	2	...	n	...

Tabela 1.6: Função f_1 para o Teorema 1.4

Se B é subconjunto de A, é possível associar cada elemento de B ao mesmo elemento de A através de uma função injetora e total f_2, conforme a Tabela 1.7.

B:	–	a_1	–	...	a_n	...
f_2 :		↓			↓	
A:	a_0	a_1	a_2	...	a_n	...

Tabela 1.7: Função f_2 para o Teorema 1.4

A composição das funções f_1 e f_2, ilustrada na Tabela 1.8, mostra que existe uma função injetora e total de B para $\mathbb{N}$.

Logo, $|B| \leq |\mathbb{N}|$, ou seja, $|B| \leq \aleph_0$. Em outras palavras, qualquer subconjunto (finito ou infinito) de um conjunto enumerável é também um conjunto enumerável. ■

B:	–	a_1	–	...	a_n	...
f_2 :		↓			↓	
A:	a_0	a_1	a_2	...	a_n	...
f_1 :	↓	↓	↓		↓	
$\mathbb{N}$	0	1	2	...	n	...

Tabela 1.8: Composição de f_1 com f_2 para o Teorema 1.4

Teorema 1.5 $(|A \cup B|, |A| = \aleph_0, |B| = \aleph_0)$ *Sejam A e B dois conjuntos quaisquer. Se $|A| = \aleph_0$ e $|B| = \aleph_0$, então $|A \cup B| = \aleph_0$.*

Justificativa Se A e B são conjuntos enumeráveis (finitos ou infinitos), então seus elementos podem ser ordenados da seguinte forma:

$$A: \quad a_0, a_1, a_2, a_3, a_4, ...a_{n-1}, a_n, a_{n+1}...$$
$$B: \quad b_0, b_1, b_2, b_3, b_4, ...b_{n-1}, b_n, b_{n+1}...$$

A enumeração dos elementos de $A \cup B$ pode ser feita através do seguinte procedimento:

$$A \cup B : a_0, b_0, a_1, b_1, a_2, b_2, ...a_{n-1}, b_{n-1}, a_n, b_n, a_{n+1}, b_{n+1}, ...$$

Portanto, $A \cup B$ é enumerável e $|A \cup B| = \aleph_0$. Em outras palavras, a união de dois conjuntos enumeráveis é sempre um conjunto enumerável. ■

Teorema 1.6 $(|A \cap B|, |A| = \aleph_0, |B| = \aleph_0)$ *Sejam A e B dois conjuntos quaisquer. Se $|A| = \aleph_0$ e $|B| = \aleph_0$, então $|A \cap B| = \aleph_0$.*

Justificativa Se $A \subseteq B$, então $A \cap B = A$ e $|A \cap B| = |A| = \aleph_0$ por hipótese. Se, por outro lado, $B \subseteq A$, então $A \cap B = B$ e $|A \cap B| = |B| = \aleph_0$ por hipótese. Finalmente, se nenhuma dessas duas condições for verdadeira, então $A \cap B \subseteq A$ e, pelo Teorema 1.4, $|A \cap B| = \aleph_0$. Portanto, em qualquer caso que se considere, $|A \cap B| = \aleph_0$. ■

Teorema 1.7 $(|A - B|, B \subseteq A, |A| = \aleph_1, |B| = \aleph_0)$ *Sejam A e B dois conjuntos, $B \subseteq A$. Se $|A| = \aleph_1$ e $|B| = \aleph_0$, então $|A - B| = \aleph_1$.*

Justificativa Suponha-se que $|A - B| = \aleph_0$. Então, de acordo com o Teorema 1.5, $|(A - B) \cup B| = \aleph_0$, o que contradiz a hipótese de que $|A| = \aleph_1$, pois $(A - B) \cup B = A$. Como $B \subseteq A$, e, portanto, $|B| \leq |A|$, conclui-se que $|A - B| = \aleph_1$. ■

2 Conceitos Básicos de Linguagens

Este capítulo apresenta e discute os principais conceitos básicos associados ao estudo de linguagens, como é o caso dos símbolos, das cadeias e das linguagens propriamente ditas, assim como dos métodos empregados para a formalização das linguagens, como é o caso das gramáticas e dos reconhecedores.

As gramáticas possuem grande importância na análise e na especificação formal da sintaxe de linguagens, principalmente das linguagens de programação, ao passo que os reconhecedores definem modelos que servem como base para a construção de analisadores léxicos e sintáticos nos compiladores e interpretadores de tais linguagens. Na prática, o estudo sistemático das linguagens formais e dos autômatos viabilizou o desenvolvimento de técnicas eficientes para a utilização econômica das linguagens de programação, originando-se em seu estudo grande parte do interesse que recai sobre os assuntos ligados a esse tema.

O capítulo se encerra com a apresentação da Hierarquia de Chomsky, usualmente empregada na classificação hierárquica das linguagens conforme seu grau de complexidade, e em torno da qual se costuma desenvolver boa parte da teoria das linguagens formais e autômatos. Tomando-se como base essa hierarquia, são efetuadas considerações preliminares sobre as propriedades e sobre a caracterização formal de cada uma das classes de linguagens por ela definidas.

2.1 Símbolos e Cadeias

Os **símbolos**,[1] também denominados **palavras** ou **átomos**, são representações gráficas, indivisíveis, empregadas na construção de **cadeias**. Estas são formadas através da justaposição de um número finito de símbolos, obtidos de algum conjunto finito não-vazio, denominado **alfabeto**.

Cada símbolo é considerado como uma unidade atômica, não importando a sua particular representação visual. São exemplos de símbolos: a, abc, $begin$, if, $5, 1024, 2.017e4$. Perceba-se que não há uma definição formal para "símbolo". Deve-se intuir o seu significado como entidade abstrata, e dessa forma aceitá-lo como base para a teoria que será desenvolvida. Pode-se dizer que se trata de um conceito primitivo.

Ao longo deste texto será adotada a seguinte convenção para denotar símbolos, cadeias e alfabetos:

- Símbolos: letras minúsculas do início do alfabeto romano: $(a, b, c...)$.
- Cadeias: letras minúsculas do final do alfabeto romano $(r, s, x, w...)$, ou letras minúsculas do alfabeto grego $(\alpha, \beta, \gamma...)$.
- Alfabetos: letras maiúsculas do alfabeto grego $(\Sigma, \Gamma, \Delta...)$.

[1] Na literatura em inglês são conhecidos como "tokens".

Exemplo 2.1 Como exemplo de alfabeto podemos mencionar o conjunto Σ dos dígitos hexadecimais, em que cada elemento (dígito) desse conjunto corresponde a um determinado símbolo:

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}$$

Naturalmente, as cadeias que podem ser construídas a partir dos símbolos desse alfabeto correspondem aos numerais hexadecimais: 123, $a0b56$, $fe5dc$, b, abc, $55efff$... □

O **comprimento** de uma cadeia é um número natural que designa a quantidade de símbolos que a compõem. O comprimento de uma cadeia α é denotado por $|\alpha|$.

Exemplo 2.2 Considerem-se as cadeias $\alpha = 1, \beta = 469, \chi = ble60, \phi = df$. Então, $|\alpha| = 1, |\beta| = 3, |\chi| = 5$ e $|\phi| = 2$. □

Dá-se o nome de **cadeia elementar** (ou **unitária**) a qualquer cadeia formada por um único símbolo, como é o caso da cadeia α do Exemplo 2.2. Naturalmente, toda cadeia unitária tem comprimento 1.

Um outro importante exemplo de alfabeto corresponde ao conjunto dos símbolos definidos em algum dicionário da língua portuguesa. Note que, diferentemente do alfabeto Σ anteriormente considerado, em que todos os símbolos eram compostos de um único caractere, os símbolos deste novo alfabeto são construídos a partir da concatenação de um número variável de caracteres (no caso, as letras do alfabeto romano). Para o presente estudo, embora representados com diversos caracteres, tais símbolos são considerados indivisíveis, e as correspondentes cadeias elementares apresentam, por essa razão, comprimento unitário quando consideradas no contexto da língua portuguesa.

Considerando-se ainda que tal alfabeto é suficientemente extenso para conter as conjugações de todos os verbos, as formas flexionadas de todos os adjetivos, substantivos etc., enfim, todas as palavras possíveis de serem empregadas em nosso idioma, então a cadeia *"Exemplo de uma cadeia no novo alfabeto"* deverá ser considerada uma cadeia válida, construída a partir dos símbolos desse alfabeto, e o seu comprimento é igual a 7. Note que esse alfabeto da língua portuguesa, apesar de extenso, é finito, e também que é possível construir uma quantidade infinita de cadeias de comprimento finito com os símbolos dele. Observe-se que, entre estas, há cadeias que na língua portuguesa não fazem sentido. Por exemplo, *"cadeia uma exemplo errado o"*. As demais são empregadas nas diversas formas de comunicação humana.

O conceito de **cadeia vazia** é especialmente importante na teoria das linguagens formais. Denota-se por ϵ a cadeia formada por uma quantidade nula de símbolos, isto é, a cadeia que não contém nenhum símbolo. Formalmente,

$$|\epsilon| = 0$$

Duas cadeias, sejam elas elementares ou não, podem ser anexadas, formando uma só cadeia, através da operação de **concatenação**. Essa operação fornece como resultado uma nova cadeia, formada pela justaposição ordenada dos símbolos que compõem os seus operandos separadamente. Observe-se que a operação de concatenação entre uma cadeia e um símbolo é realizada através da concatenação da cadeia em questão com a cadeia elementar correspondente ao símbolo.

Denota-se a concatenação de duas cadeias α e β como $\alpha \cdot \beta$ ou, simplesmente, $\alpha\beta$.

Exemplo 2.3 Considere o alfabeto $\Sigma = \{a, b, c, d\}$, e as cadeias $\alpha = abc$, $\beta = dbaca$ e $\sigma = a$.
A concatenação da cadeia α com a cadeia β é assim obtida:
$\alpha \cdot \beta = \alpha\beta = abcdbaca$, e $|\alpha\beta| = |\alpha| + |\beta| = 3 + 5 = 8$
Da mesma forma, obtém-se a concatenação de β com α:

$\beta \cdot \alpha = \beta\alpha = dbacaabc$, e, $\left|\beta\alpha\right| = \left|\beta\right| + \left|\alpha\right| = 5 + 3 = 8$
Note-se que, neste exemplo, $\alpha\beta \neq \beta\alpha$.
A concatenação da cadeia α com a cadeia elementar σ é dada por:
$\alpha \cdot \sigma = \alpha\sigma = abca$, e $\left|\alpha\sigma\right| = \left|\alpha\right| + \left|\sigma\right| = 3 + 1 = 4$
Finalmente, a concatenação da cadeia elementar σ com a cadeia β é obtida como:
$\sigma \cdot \beta = \sigma\beta = adbaca$, e $\left|\sigma\beta\right| = \left|\sigma\right| + \left|\beta\right| = 1 + 5 = 6$ □

Como se pode perceber, a operação de concatenação, embora associativa, não é comutativa. Dadas três cadeias α, β, γ quaisquer, pode-se sempre afirmar que $(\alpha\beta)\gamma = \alpha(\beta\gamma)$.

Por outro lado, dependendo dos particulares α e β considerados, pode ser que ou $\alpha\beta \neq \beta\alpha$ ou $\alpha\beta = \beta\alpha$ (por exemplo, se α e/ou β forem cadeias vazias ou, ainda, se $\alpha = \beta$).

No caso da cadeia vazia ϵ (elemento neutro em relação ao operador de concatenação) são válidas as seguintes relações:

1. $\alpha\epsilon = \epsilon\alpha = \alpha$
2. $\left|\alpha\epsilon\right| = \left|\epsilon\alpha\right| = \left|\alpha\right|$

Diz-se que uma cadeia α é um **prefixo** de outra cadeia β se for possível escrever β como $\alpha\gamma$. A cadeia α é dita **sufixo** de β se β puder ser escrita como $\gamma\alpha$. Em ambos os casos, admite-se a possibilidade de $\gamma = \epsilon$. Nos casos em que $\gamma \neq \epsilon$, diz-se que α é, respectivamente, **prefixo próprio** ou **sufixo próprio** da cadeia β. Note que a cadeia vazia ϵ pode ser considerada simultaneamente prefixo ou sufixo de qualquer cadeia.

Dadas quatro cadeias α, β, γ e δ, uma cadeia α é chamada **subcadeia** de uma cadeia β sempre que $\beta = \gamma\alpha\delta$. Note-se que, se γ ou δ ou ambos forem vazios, a definição também se aplica. Note-se também que prefixos e sufixos são casos particulares de subcadeias.

Uma cadeia α é dita o **reverso** de uma cadeia β, denotando-se o fato por $\alpha = \beta^R$, se α contiver os mesmos símbolos que β, porém justapostos no sentido inverso, ou seja:
se $\alpha = \sigma_1\sigma_2...\sigma_{n-1}\sigma_n$ então $\beta = \sigma_n\sigma_{n-1}...\sigma_2\sigma_1$
Por definição, $\epsilon^R = \epsilon$.

Exemplo 2.4 Considerem-se as cadeias $\alpha = 123abc$ e $\beta = d$. Então, $\alpha^R = cba321$ e $\beta^R = d$. □

Finalmente, convenciona-se que σ^i representa a cadeia formada por "i" símbolos σ concatenados. Por definição, $\sigma^0 = \epsilon$.

Exemplo 2.5 Considere-se o símbolo a. Então:

- $a^0 = \epsilon$;
- $a^1 = a$;
- $a^2 = aa$;
- $a^3 = aaa$;
- etc.

□

2.2 Linguagens

Uma **linguagem formal** é um conjunto, finito ou infinito, de cadeias de comprimento finito, formadas pela concatenação de elementos de um alfabeto finito e não-vazio. Além

das operações previamente definidas para conjuntos, como união, diferença, intersecção etc., outras operações, tais como a concatenação e os fechamentos, também são fundamentais para a definição e o estudo das linguagens formais.

Antes de apresentá-las, convém notar a distinção que há entre os seguintes conceitos: cadeia vazia ϵ, conjunto vazio $\varnothing$ e o conjunto que contém apenas a cadeia vazia $\{\epsilon\}$.

O primeiro deles, ϵ, denota a **cadeia** vazia, ou seja, uma cadeia de comprimento zero, ao passo que os dois seguintes são casos particulares de **linguagens** (que por sua vez são conjuntos): $\varnothing$ denota uma linguagem vazia, ou seja, uma linguagem que não contém nenhuma cadeia, e $\{\epsilon\}$ denota uma linguagem que contém uma única cadeia, a cadeia vazia. Observe-se que $\left|\varnothing\right| = 0$ e $\left|\{\epsilon\}\right| = 1$.

Note-se a diferença conceitual que há entre alfabetos, linguagens e cadeias. Alfabetos são conjuntos, finitos e não-vazios, de símbolos, através de cuja concatenação são obtidas as **cadeias**. Linguagens, por sua vez, são conjuntos, finitos (eventualmente vazios) ou infinitos, de cadeias. Uma cadeia é também denominada **sentença** de uma linguagem, ou simplesmente sentença, no caso de ela pertencer à linguagem em questão. Linguagens são, portanto, coleções de sentenças sobre um dado alfabeto.

A Figura 2.1 ilustra a relação entre os conceitos de símbolo, alfabeto, cadeia e linguagem:

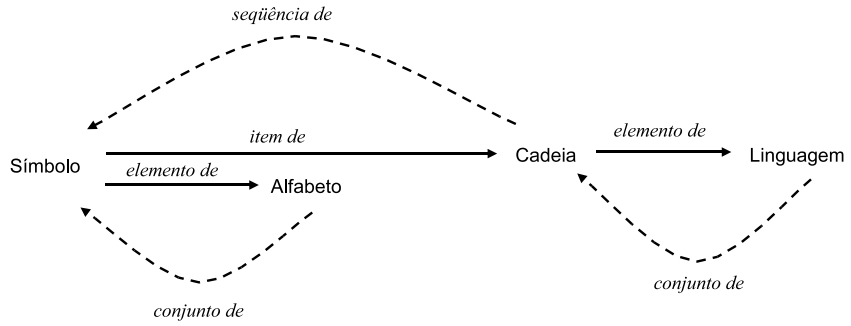

Figura 2.1: Símbolo, alfabeto, cadeia e linguagem

As várias leituras contidas na Figura 2.1 são: "símbolo é elemento de alfabeto"; "alfabeto é conjunto de símbolos"; "símbolo é item de cadeia"; "cadeia é seqüência de símbolos"; "cadeia é elemento de linguagem" e "linguagem é conjunto de cadeias".

Outra maneira de associar significados aos termos "símbolo", "alfabeto", "cadeia" e "linguagem" é apresentada na Figura 2.2, que também ilustra o conceito de "sentença".

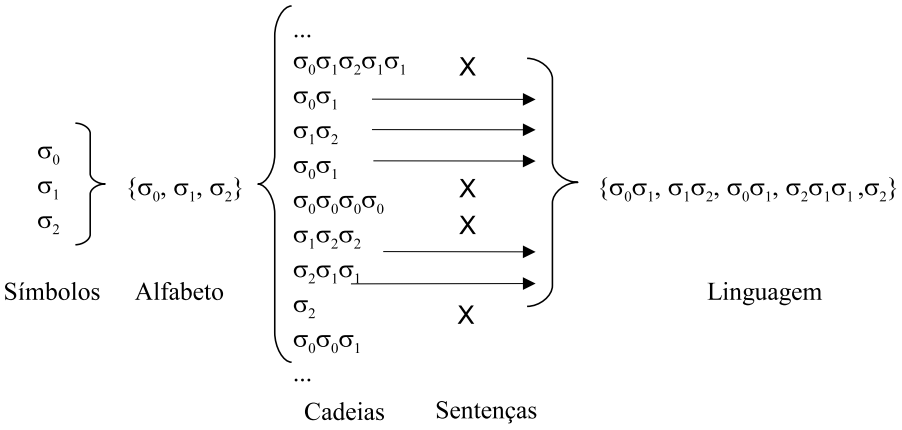

Figura 2.2: Símbolo, alfabeto, cadeia, sentença e linguagem

A Figura 2.2 facilita o entendimento das relações entre os conceitos: (i) um conjunto de símbolos forma um alfabeto, (ii) a partir de um alfabeto (finito) formam-se (infinitas) cadeias; (iii) determinadas cadeias são escolhidas para fazer parte de uma linguagem; (iv) uma linguagem é um conjunto de cadeias, que por isso são também denominadas sentenças.

Exemplo 2.6 O símbolo a é elemento do alfabeto $\{a\}$ e também um item da cadeia aaa, que por sua vez é elemento da linguagem $\{aaa\}$. Por outro lado, a linguagem $\{aaa\}$ é um conjunto que contém a cadeia aaa, a cadeia aaa é uma seqüência de símbolos a e o alfabeto $\{a\}$ contém o símbolo a. A Figura 2.3 ilustra esses conceitos, conforme a Figura 2.1.

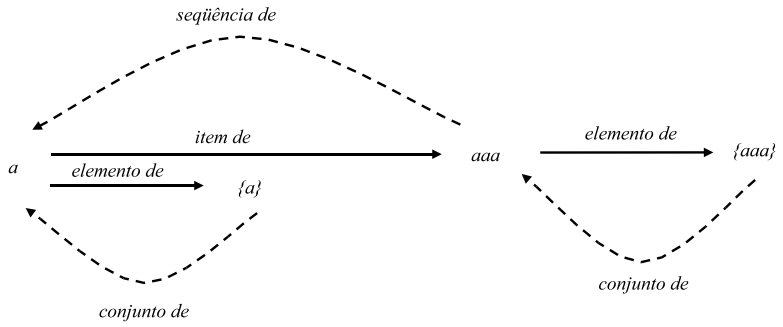

Figura 2.3: $a, \{a\}, aaa, \{aaa\}$

□

Exemplo 2.7 A Figura 2.4 ilustra uma aplicação dos conceitos da Figura 2.2 ao alfabeto $\{a, b\}$. A linguagem apresentada é, naturalmente, apenas uma das inúmeras que podem ser criadas a partir desse alfabeto.

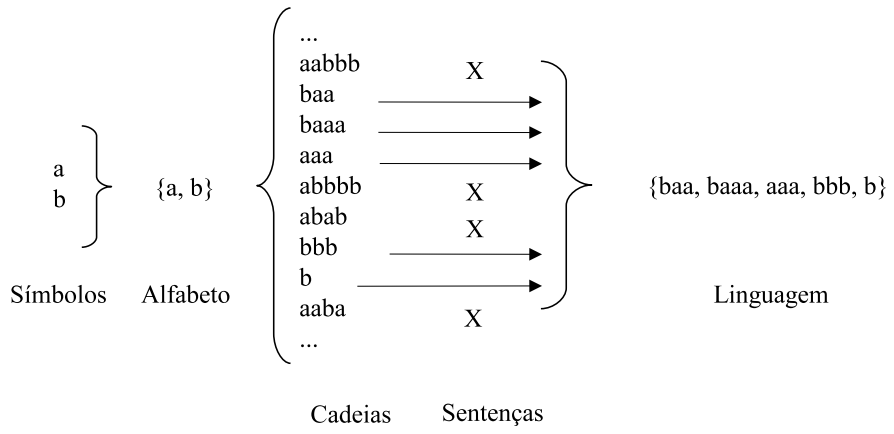

Figura 2.4: a, b, cadeias, sentenças e linguagem

□

A **concatenação** de duas linguagens X e Y, denotada por $X \cdot Y$ ou simplesmente XY, corresponde a um conjunto Z formado pela coleção de todas as cadeias que possam ser obtidas pela concatenação de cadeias $x \in X$ com cadeias $y \in Y$, nesta ordem. Formalmente,

$$Z = X \cdot Y = XY = \{xy \mid x \in X \text{ e } y \in Y\}$$

A concatenação $\Sigma\Sigma$, que gera cadeias de comprimento 2 formadas sobre um alfabeto Σ, é também representada por Σ^2. Analogamente, a concatenação $\Sigma\Sigma\Sigma$, que gera cadeias de comprimento 3 sobre o alfabeto Σ, é representada como Σ^3, e assim sucessivamente. Generalizando-se:

$$\Sigma^i = \Sigma\Sigma^{i-1}, i \geq 0$$

Por definição, $\Sigma^0 = \{\epsilon\}$

Exemplo 2.8 Considere-se $\Sigma = \{a, b, c\}$. Então,

$$\begin{aligned} \Sigma^0 &= \{\epsilon\} \\ \Sigma^1 &= \{a, b, c\} \\ \Sigma^2 &= \{aa, ab, ac, ba, bb, bc, ca, cb, cc\} \\ \Sigma^3 &= \{aaa, aab, aac, aba, abb, abc, ..., ccc\} \\ \text{etc.} & \end{aligned}$$

□

O **fechamento reflexivo e transitivo** (às vezes chamado **fechamento recursivo e transitivo**) de um alfabeto Σ é definido como o conjunto (infinito) que contém todas as possíveis cadeias que podem ser construídas sobre o alfabeto dado, incluindo a cadeia vazia. Esse conjunto, denotado por Σ^*, contém, naturalmente, todas as cadeias que podem ser definidas sobre o alfabeto Σ. Formalmente, o fechamento reflexivo e transitivo de um conjunto Σ é definido como:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ... = \bigcup_{i=0}^{\infty} \Sigma^i$$

Em outras palavras, o fechamento reflexivo e transitivo de um alfabeto é a união de todos os conjuntos que representam as possíveis cadeias de comprimentos zero, um, dois,

três, e assim por diante, ou seja, é a coleção de todas as cadeias, de qualquer comprimento, que possam ser formadas por concatenação a partir dos símbolos do alfabeto.

A definição de uma linguagem pode, portanto, ser formulada de maneira mais rigorosa com o auxílio da operação de fechamento reflexivo e transitivo: sendo uma linguagem qualquer coleção de cadeias sobre um determinado alfabeto Σ, e como Σ^* contém todas as possíveis cadeias sobre Σ, então toda e qualquer linguagem L sobre um alfabeto Σ sempre poderá ser definida como sendo um subconjunto de Σ^*, ou seja, $L \subseteq \Sigma^*$.

Diz-se que a **maior** linguagem que se pode definir sobre um alfabeto Σ, observando-se um conjunto qualquer P de propriedades, corresponde ao conjunto de **todas** as cadeias $w \in \Sigma^*$ tais que w satisfaz simultaneamente a **todas** as propriedades $p_i \in P$.

De uma forma geral, sempre que for feita uma referência a uma determinada linguagem L cujas cadeias satisfaçam a um certo conjunto de propriedades P, estará implícita (a menos de ressalva em contrário) a condição de que se trata da maior linguagem definida sobre L, cujas cadeias satisfaçam o conjunto de propriedades P.

Um caso particular importante a se considerar é a linguagem cujo conjunto P de propriedades seja o menos restritivo possível, considerando toda e qualquer cadeia de qualquer comprimento (finito) como sendo válida. Assim, a maior linguagem dentre todas as que podem ser definidas sobre um alfabeto Σ qualquer é $L = \Sigma^*$ (note-se, neste caso, que a única propriedade a ser satisfeita pelas cadeias de L é que elas "sejam definidas sobre Σ", ou seja, obtidas a partir da simples justaposição de símbolos de Σ). Qualquer outra linguagem definida sobre esse mesmo alfabeto corresponderá obrigatoriamente a um subconjunto (eventualmente próprio) de Σ^*.

Por outro lado, a **menor** linguagem que pode ser definida sobre um alfabeto Σ qualquer é $\varnothing$, ou seja, a linguagem vazia ou a linguagem composta por zero sentenças.

Finalmente, como o conjunto de todos os subconjuntos possíveis de serem obtidos a partir de Σ^* é 2^{Σ^*}, tem-se que 2^{Σ^*} representa o conjunto de **todas** as linguagens que podem ser definidas sobre o alfabeto Σ.

Em resumo:

- $\varnothing$ é o conjunto constituído por zero cadeias e corresponde à menor linguagem que se pode definir sobre um alfabeto Σ qualquer;
- Σ^* é o conjunto de todas as cadeias possíveis de serem construídas sobre Σ e corresponde à maior de todas as linguagens que pode ser definida sobre Σ;
- 2^{Σ^*} é o conjunto de todos os subconjuntos possíveis de serem obtidos a partir de Σ^*, e corresponde ao conjunto formado por todas as possíveis linguagens que podem ser definidas sobre Σ. Observe-se que $\varnothing \in 2^{\Sigma^*}$, e também que $\Sigma^* \in 2^{\Sigma^*}$.

Exemplo 2.9 Seja $\Sigma = \{a, b, c\}$ e P o conjunto formado pela única propriedade "todas as cadeias são iniciadas com o símbolo a". Então:

- A linguagem $L_0 = \varnothing$ é a menor linguagem que pode ser definida sobre Σ;
- A linguagem $L_1 = \{a, ab, ac, abc, acb\}$ é finita e observa P;
- A linguagem $L_2 = \{a\}\{a\}^*\{b\}^*\{c\}^*$ é infinita e observa P;
- A linguagem $L_3 = \{a\}\{a, b, c\}^*$ é infinita, observa P e, dentre todas as que observam P, trata-se da maior linguagem, pois não existe nenhuma outra cadeia em Σ^* que satisfaça a P e não pertença a L_3;
- $L_0 \subseteq \Sigma^*$, $L_1 \subseteq \Sigma^*$, $L_2 \subseteq \Sigma^*$, $L_3 \subseteq \Sigma^*$;

- $L_0 \in 2^{\Sigma^*}$, $L_1 \in 2^{\Sigma^*}$, $L_2 \in 2^{\Sigma^*}$, $L_3 \in 2^{\Sigma^*}$;
- Além de L_0, L_1, L_2 e L_3, existem inúmeras outras linguagens que podem ser definidas sobre Σ.

□

O **fechamento transitivo** de um alfabeto Σ, denotado por Σ^+, é definido de maneira análoga ao fechamento reflexivo e transitivo, diferindo deste apenas por não incluir o conjunto Σ^0:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ... = \bigcup_{i=1}^{\infty} \Sigma^i$$

Como se pode perceber, $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$. Observe-se ainda que, embora aparentemente seja conseqüência da anterior, a afirmação $\Sigma^+ = \Sigma^* - \{\epsilon\}$ só será válida nos casos em que Σ não contiver a cadeia vazia.

Exemplo 2.10 Seja $\Sigma = \{n, (,), +, *, -, /\}$. Neste caso:

- $\Sigma^* = \{\epsilon, n, n+n, -n), */, n()), n-(n*n), ...\}$
- $\Sigma^+ = \{n, n+n, -n), */, n()), n-(n*n), ...\}$
- $\Sigma^+ = \Sigma^* - \{\epsilon\}$, pois $\epsilon \notin \Sigma$

□

A **complementação** de uma linguagem X definida sobre um alfabeto Σ é definida como:

$$\overline{X} = \Sigma^* - X$$

Diz-se que uma linguagem exibe a propriedade do **prefixo (sufixo) próprio** sempre que não houver nenhuma cadeia a ela pertencente que seja prefixo (sufixo) próprio de outra cadeia dessa mesma linguagem. Formalmente:

- Prefixo próprio: não existe $\alpha \in L \mid \beta \neq \epsilon$ e $\alpha\beta \in L$
- Sufixo próprio: não existe $\alpha \in L \mid \beta \neq \epsilon$ e $\beta\alpha \in L$

Exemplo 2.11 Considere as seguintes linguagens:

$$\begin{aligned} L_1 &= \{a^i b^i \mid i \geq 1\} = \{ab, aabb, aaabbb, ...\} \\ L_2 &= \{ab^i \mid i \geq 1\} = \{ab, abb, abbb, abbbb, ...\} \end{aligned}$$

Neste exemplo, a linguagem L_1 exibe a propriedade do prefixo próprio, ao passo que a linguagem L_2 não a exibe. A propriedade do sufixo próprio é exibida por ambas as linguagens. □

Diz-se que uma linguagem L_1 é o **reverso** de uma linguagem L_2, denotando-se o fato por $L_1 = L_2^R$ (ou $L_2 = L_1^R$), quando as sentenças de L_1 corresponderem ao reverso das sentenças de L_2. Formalmente:

$$L_1 = L_2^R = \{x^R \mid x \in L_2\}$$

Exemplo 2.12 Seja $L_2 = \{\epsilon, a, ab, abc\}$. Então, $L_1 = L_2^R = \{\epsilon, a, ba, cba\}$. □

Define-se o **quociente** de uma linguagem L_1 por uma outra linguagem L_2, denotado por L_1/L_2, como sendo a linguagem:

$$L_1/L_2 = \{x \mid xy \in L_1 \text{ e } y \in L_2\}$$

Exemplo 2.13 Considerem-se as linguagens:

$$\begin{aligned} L_1 &= \{a^i b \mid i \geq 0\} \\ L_2 &= \{a^i bc^i \mid i \geq 0\} \\ L_3 &= \{b\} \\ L_4 &= \{a^i b \mid i \geq 1\} \\ L_5 &= \{bc^i \mid i \geq 0\} \\ L_6 &= \{c^i b \mid i \geq 0\} \\ L_7 &= \{a^i \mid i \geq 0\} \end{aligned}$$

Então, são verdadeiras as seguintes relações:

- $L_1/L_3 = L_7$:

  $L_1 = \{b, ab, aab, aaab...\}$

  $L_3 = \{b\}$

  Para cada uma das cadeias pertencentes a L_1, é possível identificar o (único) membro de L_3 como sufixo dos membros de L_1. Logo, através da remoção desse sufixo b, de todas as cadeias de L_1, o conjunto resultante será L_7.

- $L_1/L_4 = L_7$:

  $L_1 = \{b, ab, aab, aaab...\}$

  $L_4 = \{ab, aab, aaab...\}$

  Nenhuma das cadeias pertencentes a L_4 pode ser considerada sufixo de $b \in L_1$. Por outro lado, $ab \in L_4$ é sufixo de $ab \in L_1$, de $aab \in L_1$ etc., resultando que $\epsilon, a, aa...$ pertencem ao quociente de L_1 por L_4. O mesmo ocorre com as cadeias $aab, aaab...$ pertencentes a L_4, sem no entanto modificar o resultado da operação.

- $L_5/L_7 = \varnothing$

  $L_5 = \{b, bc, bcc, bccc...\}$

  $L_7 = \{ab, aab, aaab...\}$

  Como nenhuma cadeia de L_5 contém o símbolo a (presente em todas as cadeias de L_7), conclui-se não ser possível representar nenhuma cadeia de L_5 através da concatenação de um prefixo com uma cadeia de L_7.

- $L_2/L_6 = L_7$

  $L_2 = \{b, ab, bc, abc, aab, bcc, aabc, abcc, aabcc...\}$

  $L_6 = \{b, cb, ccb, cccb...\}$

  Como nenhum dos elementos de L_6 tem o símbolo c como sufixo, todas as cadeias pertencentes a L_2 que terminam com o símbolo c não possuem sufixo em L_6. No entanto, todas as cadeias de L_2 compostas apenas por símbolos a e b possuem a cadeia $b \in L_6$ como sufixo. Removido de tais cadeias esse sufixo, restam as cadeias formadas por zero ou mais símbolos a, ou seja, a linguagem L_7.

- $L_2/L_1 = L_7$

  $L_2 = \{b, ab, bc, abc, aab, bcc, aabc, abcc, aabcc...\}$

  $L_1 = \{b, ab, aab, aaab...\}$

Nenhuma das cadeias da linguagem L_2 que termina com o símbolo c possui sufixo em L_1, uma vez que nenhuma das cadeias dessa linguagem termina com o símbolo c. Por outro lado, todas as cadeias de L_2 compostas apenas pelos símbolos a e b possuem como sufixo em L_1 as cadeias de mesmo formato, porém com uma quantidade menor ou igual de símbolos a (por exemplo, $aab \in L_2$ possui como sufixo as cadeias b, ab, aab), resultando que o quociente é formado por cadeias com uma quantidade maior ou igual a zero de símbolos a (no exemplo, aa, a, ϵ).

- $L_5/L_2 = \{\epsilon\}$

  $L_5 = \{b, bc, bcc, bccc...\}$

  $L_2 = \{b, ab, bc, abc, aab, bcc, aabc, abcc, aabcc...\}$

  Não é difícil perceber, neste caso, que para a cadeia $b \in L_5$, $b \in L_2$ é o único sufixo da mesma em L_2 e, portanto, $\epsilon \in L_5/L_2$. O mesmo raciocínio vale para a cadeia $bc \in L_5$, e assim por diante. Logo, o quociente corresponde à linguagem formada pela única cadeia ϵ. □

Uma **substituição** é uma função que mapeia os elementos de um alfabeto Σ_1 em linguagens sobre um alfabeto Σ_2. Formalmente:

$$s : \Sigma_1 \rightarrow 2^{\Sigma_2^*}$$

Diz-se que uma substituição é um **homomorfismo** se $2^{\Sigma_2^*}$ contiver apenas um elemento para cada $\sigma \in \Sigma_1$. Neste caso, costuma-se considerar os elementos do conjunto imagem como simples cadeias e não como conjuntos compostos por um único elemento:

$$h : \Sigma_1 \rightarrow \Sigma_2^*$$

Assim, um homomorfismo é uma função que mapeia cada símbolo de um alfabeto Σ_1 em uma cadeia única contida em uma linguagem Σ_2^*.

Substituições permitem a especificação de transliterações de linguagens, nas quais os símbolos do alfabeto da linguagem original por cadeias de símbolos de um novo alfabeto.

O domínio de uma substituição pode ser estendido para Σ_1^* da seguinte forma:

- $s(\epsilon) = \epsilon$
- $s(a\alpha) = s(a)s(\alpha), a \in \Sigma_1, \alpha \in \Sigma_1^*$

Em outras palavras, uma substituição é um mapeamento que preserva a ordem dos componentes das sentenças. Assim, a substituição de uma linguagem L é definida como:

$$s(L) = \{y \mid y = s(x) \text{ para } x \in L\}$$

Exemplo 2.14 Considere-se a linguagem $L = \{a^i b^i c^i \mid i \geq 1\}$ e a substituição s sobre $\Sigma = \{x, y, z\}$:

- $s(a) = \{x\}$;
- $s(b) = \{y, yy\}$;
- $s(c) = \{z, zz, zzz\}$.

A aplicação de s em L define a linguagem $s(L) = \{x^i y^j z^k \mid i \geq 1, i \leq j \leq 2i, i \leq k \leq 3i\}$.

A título de ilustração, serão apresentadas a seguir todas as transliterações possíveis para a cadeia abc obtidas através da substituição s acima definida:

$$s(abc) = \{xyz, xyzz, xyzzz, xyyz, xyyzzz\}$$

Note-se que, neste exemplo, $s(abc) = s(a)s(b)s(c)$, onde:

$s(a)$ pode ser substituído apenas por x
$s(b)$ pode ser substituído por y ou yy
$s(c)$ pode ser substituído por z, zz ou zzz

O conjunto $s(abc)$ é obtido a partir de todas as combinações possíveis de substituições para os símbolos a, b e c na cadeia abc conforme s. □

Exemplo 2.15 Considere-se agora a mesma linguagem L do Exemplo 2.14 e o homomorfismo h sobre o respectivo alfabeto Σ:

- $h(a) = x$;
- $h(b) = x$;
- $h(c) = z$.

A linguagem definida por esse homomorfismo é $h(L) = \{x^{2i}z^i \mid i \geq 1\}$. □

Nos casos em que h é um homomorfismo, diz-se que $h(L)$ é a **imagem homomórfica** de L. A **imagem homomórfica inversa** de uma cadeia y é definida através de um **homomorfismo inverso** h^{-1}:

$$h^{-1}(y) = \{x \mid y = h(x)\}$$

Observe-se que a imagem homomórfica inversa de uma cadeia é definida como um conjunto de cadeias, e não como uma cadeia única, uma vez que a função h não é necessariamente injetora. A imagem homomórfica inversa de uma linguagem L é definida como:

$$h^{-1}(L) = \{x \mid h(x) \in L\}$$

Exemplo 2.16 Ainda no Exemplo 2.3, a imagem homomórfica inversa de $h(L)$ pode ser definida através do homomorfismo inverso h^{-1}:

- $h^{-1}(x) = \{a, b\}$;
- $h^{-1}(z) = \{c\}$

Pelo fato de a função h originalmente adotada não ser usualmente injetora, obtém-se:

$$h^{-1}(h(L)) = \{(a \mid b)^{2i}c^i \mid i \geq 1\}$$

e, portanto, $h^{-1}(h(L)) \neq L$. □

Um homomorfismo é denominado **isomorfismo** sempre que a função h for injetora. Neste caso, $h(L)$ é denominada **imagem isomórfica** de L. Isomorfismos viabilizam a definição de imagens homomórficas inversas através de funções injetoras denominadas **isomorfismos inversos**. Neste caso, a linguagem $h^{-1}(L)$ é denominada **imagem isomórfica inversa** de L.

Exemplo 2.17 Considere-se a linguagem $L_1 = \{a^i b^i \mid i \geq 1\}$ sobre o alfabeto $\Sigma_1 = \{a, b\}$ e o isomorfismo j, definido sobre $\Sigma_2 = \{c, d\}$:

- $j(a) = c$;
- $j(b) = dd$.

Neste caso, a imagem isomórfica de L_1, ou seja, a linguagem L_2, resultante da aplicação do isomorfismo j sobre o alfabeto da linguagem L_1, é a seguinte:

$$L_2 = \{c^i d^{2i} \mid i \geq 1\}$$

A imagem isomórfica inversa de L_2, ou seja, a linguagem L_1, pode ser obtida a partir do isomorfismo inverso:

- $j^{-1}(c) = a$;
- $j^{-1}(dd) = b$.

Observe, neste caso, que $j^{-1}(j(L_1)) = L_1$. □

Isomorfismos e isomorfismos inversos são muito úteis, pois permitem mudar de Σ_1 para Σ_2 o domínio na resolução de problemas, visando com isso facilitar a sua resolução no domínio Σ_2. Por se tratar de um mapeamento feito através de uma função injetora, torna-se sempre possível retornar ao domínio original Σ_1, preservando-se a associação unívoca entre os elementos dos domínios considerados.

Exemplo 2.18 Considere-se $L = \mathbb{N}$ e o isomorfismo j, sobre $\Sigma = \{0, 1\}$:

- $j(0) = 0000$
- $j(1) = 0001$
- $j(2) = 0010$
- $j(3) = 0011$
- $j(4) = 0100$
- $j(5) = 0101$
- $j(6) = 0110$
- $j(7) = 0111$
- $j(8) = 1000$
- $j(9) = 1001$

$j(L)$ mapeia os números naturais na representação equivalente em BCD (*Binary Coded Decimal*). O isomorfismo inverso j^{-1}, abaixo, permite retornar ao domínio dos números naturais:

- $j^{-1}(0000) = 0$
- $j^{-1}(0001) = 1$
- $j^{-1}(0010) = 2$
- $j^{-1}(0011) = 3$
- $j^{-1}(0100) = 4$
- $j^{-1}(0101) = 5$
- $j^{-1}(0110) = 6$
- $j^{-1}(0111) = 7$

- $j^{-1}(1000) = 8$
- $j^{-1}(1001) = 9$

□

Conforme já mencionado, o conjunto Σ^* contém todas as possíveis linguagens que podem ser definidas sobre o alfabeto Σ. Na prática, as linguagens de interesse costumam ser definidas como um subconjunto próprio de Σ^*, para um dado alfabeto Σ.

Exemplo 2.19 Considere-se o alfabeto $\Sigma_1 = \{n, (,), +, *, -, /\}$. Uma linguagem L_1, passível de ser definida sobre o alfabeto Σ_1, e que corresponde a um subconjunto próprio de Σ_1^*, é aquela em que as cadeias se assemelham, do ponto vista estrutural, às cadeias que representam expressões aritméticas bem-formadas em muitas linguagens populares de programação de alto nível.

A seguir estão relacionadas algumas das cadeias que fazem parte de L_1, de acordo com esse critério:

- n
- n+n
- (n*n)
- n*(n(n+n+n))

Portanto, $L_1 = \{n, n+n, (n*n), n*(n(n+n+n)), ...\} \subset \Sigma_1^*$. São exemplos de cadeias pertencentes a $\Sigma_1^* - L_1$:

- n++
- nn*
- (n*n)))
- n*-(n(+n+/))
- ϵ

□

As linguagens de interesse prático, como é o caso das linguagens de programação, normalmente correspondem a um subconjunto próprio do fechamento reflexivo e transitivo do alfabeto sobre o qual as suas cadeias são construídas. Assim, tornam-se necessários métodos e notações que permitam, dentro do conjunto fechamento, identificar as cadeias que efetivamente pertencem à linguagem que estiver sendo definida, descartando-se as demais.

Um outro aspecto importante, referente à definição rigorosa das linguagens formais, diz respeito ao fato de que, em sua maioria, as linguagens de interesse contêm, se não uma quantidade infinita, ao menos um número finito, porém muito grande, de cadeias.

Por esses dois motivos, há um interesse muito grande em relação a métodos que permitam especificar linguagens, sejam elas finitas ou não, através de representações finitas. Por outro lado, nem todas as linguagens podem ser representadas por meio de uma especificação finita. Apesar do reduzido interesse prático que recai sobre tais linguagens, é possível comprovar a existência de linguagens para as quais é impossível se obter uma representação finita.

Neste texto serão considerados três métodos dentre os mais empregados para a representação finita de linguagens:

1. **Gramáticas**: correspondem a especificações finitas de dispositivos de geração de cadeias. Um dispositivo desse tipo deve ser capaz de gerar toda e qualquer cadeia

pertencente à linguagem definida pela gramática, e nada mais. Assim, as cadeias não pertencentes à linguagem não devem poder ser geradas pela gramática em questão. Essa forma de especificação é aplicável para linguagens finitas e infinitas.

2. **Reconhecedores**: correspondem a especificações finitas de dispositivos de aceitação de cadeias. Um dispositivo desse tipo deverá aceitar toda e qualquer cadeia pertencente à linguagem por ele definido, e rejeitar todas as cadeias não-pertencentes à linguagem. O método é aplicável para a especificação formal de linguagens finitas e infinitas.

3. **Enumerações**: relacionam, de forma explícita e exaustiva, todas as cadeias pertencentes à particular linguagem a ser especificada. Toda e qualquer cadeia pertencente à linguagem deve constar desta relação. As cadeias não pertencentes à linguagem não fazem parte dessa relação. Aplicam-se apenas para a especificação de linguagens finitas e preferencialmente não muito extensas.

Na prática, as gramáticas e os reconhecedores, além de oferecerem uma grande concisão na representação de linguagens de cardinalidade elevada, também podem ser empregados na definição de linguagens infinitas, ao contrário das enumerações. Em contraste com o que ocorre com as enumerações, as gramáticas e os reconhecedores geralmente possibilitam uma percepção melhor da estrutura sintática inerente às sentenças das linguagens por eles definidas.

Diz-se que uma gramática é **equivalente** a um reconhecedor se as duas seguintes condições forem simultaneamente verificadas (lembrar que os formalismos devem definir todas as sentenças da linguagem desejada, e nenhuma outra cadeia):

1. Toda cadeia gerada pela gramática é também aceita pelo reconhecedor.

2. Toda cadeia aceita pelo reconhecedor é também gerada pela gramática.

Exemplo 2.20 Considerem-se a gramática G e o reconhecedor M, respectivamente definidos através das linguagens gerada e aceita:

- $G \mid L_1(G) = \{\alpha \in \{a,b\}^* \mid$ o primeiro símbolo de α é "a"$\}$
- $M \mid L_2(M) = \{\beta \in \{a,b\}^* \mid$ o primeiro símbolo de β é "a" e o último símbolo é "b"$\}$

Portanto:

- $L_1 = \{a, aa, ab, aaa, aab, aba, abb, aaa...\}$
- $L_2 = \{ab, aab, abb, aaab, aabb, abab, abbb...\}$

É fácil perceber que a condição (2) acima é verificada, mas a condição (1) não. Por exemplo, a cadeia $ab \in L_2$ e $ab \in L_1$. Por outro lado, a cadeia $aba \in L_1$, porém $aba \notin L_2$. Logo, $L_1 \subset L_2$ e não se pode dizer que G e M sejam equivalentes. □

Conforme discutido mais adiante, as gramáticas e os reconhecedores são formas duais de representação de linguagens, ou seja, para cada gramática é possível obter um reconhecedor que aceite a linguagem correspondente e vice-versa. À particular notação utilizada para representar uma linguagem, seja através de gramáticas ou de reconhecedores, dá-se o nome de **metalinguagem**.

As Seções 2.3 e 2.5 discutem mais detalhadamente os conceitos fundamentais e as notações gerais empregadas na formalização e no estudo de gramáticas e reconhecedores.

A Seção 2.6 é dedicada ao estudo da classificação de linguagens e gramáticas, estabelecido por **Noam Chomsky**, um dos primeiros estudiosos e formuladores da teoria das linguagens formais. Nos capítulos que seguem, os aspectos de equivalência entre gramáticas e reconhecedores serão aprofundados, considerando cada classe particular de linguagens, de acordo com essa classificação.

Para finalizar o presente item, deve-se mencionar que as linguagens apresentam duas componentes básicas: sintaxe e semântica. A **sintaxe** de uma linguagem refere-se à sua apresentação visual, à forma, à estrutura de suas cadeias, e não leva em consideração qualquer informação sobre o significado associado às mesmas. O significado que se atribui a uma cadeia, ou conjunto de cadeias de uma mesma linguagem, deriva do significado que se atribui às construções da linguagem, ou seja, decorre diretamente da sua **semântica**.

Sintaxe e semântica constituem tópicos que se complementam, sendo de muito interesse e grande aplicação em computação. O presente texto trata, porém, apenas dos aspectos referentes à formalização da estrutura das linguagens, bem como do estudo de suas propriedades sintáticas. Devido à sua complexidade, muito maior que a da sintaxe, o estudo da semântica formal das linguagens está bem menos desenvolvido. No entanto, consideráveis progressos teóricos e práticos vêm sendo obtidos nas duas últimas décadas em semântica formal, configurando tema de estudos avançados na área das linguagens formais ([59], [30]). No entanto, o tema escapa ao escopo deste texto.

2.3 Gramáticas

Também conhecidas como **dispositivos generativos**, **dispositivos de síntese**, ou ainda dispositivos de geração de cadeias, as **gramáticas** constituem sistemas formais baseados em regras de substituição, através dos quais é possível sintetizar, de forma exaustiva, o conjunto das cadeias que compõem uma determinada linguagem.

Para ilustrar esse conceito, nada melhor do que a própria noção intuitiva, adquirida à época do ensino fundamental, do significado do termo "gramática": o livro através do qual são aprendidas as regras que indicam como falar e escrever corretamente um idioma.

Como se sabe, as regras assim definidas especificam combinações válidas dos símbolos que compõem o alfabeto — os diversos verbos, substantivos, adjetivos, advérbios, pronomes, artigos etc. —, e isso é feito com o auxílio de entidades abstratas denominadas classes sintáticas: sujeito, predicado etc. Assim, por exemplo, a frase "O menino atravessou a rua distraidamente" é considerada correta, do ponto de vista gramatical, pois ela obedece a uma das inúmeras regras de formação de frases, baseada no padrão sujeito + predicado + complemento.

De acordo com tais regras, um sujeito pode ser composto por um *artigo* (**"O"**) seguido de um *substantivo* (**"menino"**), o *predicado* pode conter um *verbo* (**"atravessou"**) e um correspondente *objeto direto* (**"a rua"**), e o *complemento* pode modificar a ação (**"distraidamente"**). O *objeto direto*, por sua vez, pode seguir o mesmo padrão estrutural do *sujeito*: *artigo* (**"a"**) seguido de *substantivo* (**"rua"**).

Naturalmente, o conjunto das regras que formam uma "gramática" deve ser suficiente para permitir a elaboração de qualquer frase ou discurso corretamente construído em um determinado idioma, e não deve permitir a construção de qualquer cadeia que não pertença à linguagem. Convém notar, no exemplo do parágrafo acima, que os termos em *itálico* correspondem às denominadas classes sintáticas do português, e os termos em **negrito**, aos símbolos que efetivamente fazem parte do seu alfabeto.

Assim como ocorre no caso das linguagens naturais, as linguagens formais também podem ser especificadas através de "gramáticas" a elas associadas. No caso das gramá-

ticas das linguagens formais, que constituem o objeto deste estudo, a analogia com as "gramáticas" das linguagens naturais é muito grande. Tratam-se, as primeiras, de conjuntos de regras que, quando aplicadas de forma recorrente, possibilitam a geração de todas as cadeias pertencentes a uma determinada linguagem.

Diferentemente das gramáticas das linguagens naturais, que são descritas por intermédio de linguagens também naturais (muitas vezes a mesma que está sendo descrita pela gramática), as gramáticas das linguagens formais são descritas utilizando notações matemáticas rigorosas que visam, entre outros objetivos, evitar dúvidas na sua interpretação. Tais notações recebem a denominação de **metalinguagens** — linguagens que são empregadas para definir outras linguagens.

Formalmente, uma gramática G pode ser definida como sendo uma quádrupla[2]:

$$G = (V, \Sigma, P, S)$$

onde:

- V é o **vocabulário** da gramática; corresponde a um conjunto (finito e não-vazio) de símbolos;
- Σ é o conjunto (finito e não-vazio) dos símbolos **terminais** da gramática; também denominado **alfabeto**;
- P é o conjunto (finito e não-vazio) de **produções** ou **regras de substituição** da gramática;
- S é a **raiz** da gramática, $S \in V$.

Adicionalmente, define-se $N = V - \Sigma$ como sendo o conjunto dos símbolos **não-terminais** da gramática. Σ corresponde ao conjunto dos símbolos que podem ser justapostos para compor as sentenças da linguagem que se está definindo, e N corresponde ao conjunto dos símbolos intermediários (classes sintáticas) utilizados na estruturação e na geração de sentenças, sem no entanto fazer parte das mesmas. Σ, N e P são conjuntos finitos e não-vazios. P é o conjunto das produções gramaticais, que obedecem à forma geral:

$$\alpha \rightarrow \beta, \quad \text{com } \alpha \in V^*NV^* \text{ e } \beta \in V^*$$

Em outras palavras, α é uma cadeia qualquer constituída por elementos de V, contendo pelo menos um símbolo não-terminal, e β é uma cadeia qualquer, eventualmente vazia, de elementos de V.

De fato, "$\rightarrow$" é uma relação sobre os conjuntos V^*NV^* e V^*, uma vez que:

$$P = \{(\alpha, \beta) \mid (\alpha, \beta) \in V^*NV^* \times V^*\}$$

ou seja, P é um subconjunto de $V^*NV^* \times V^*$.

Exemplo 2.21 Seja $G_1 = (V_1, \Sigma_1, P_1, S)$, com:

[2]Dependendo da preferência dos autores, as gramáticas podem ser definidas de outras maneiras, modificando-se a ordem dos componentes, ou explicitando/implicitando determinados conceitos. Ao ler textos sobre este assunto, é importante verificar antes qual foi a notação adotada pelo autor, para que se possa interpretá-la corretamente.

$$\begin{aligned} V_1 &= \{0, 1, 2, 3, S, A\} \\ \Sigma_1 &= \{0, 1, 2, 3\} \\ N_1 &= \{S, A\} \\ P_1 &= \{S \to 0S33, S \to A, A \to 12, A \to \epsilon\} \end{aligned}$$

É fácil verificar que G_1 está formulada de acordo com as regras gerais acima enunciadas para a especificação de gramáticas. □

Denomina-se **forma sentencial** qualquer cadeia obtida pela aplicação recorrente das seguintes regras de substituição:

1. S (a raiz da gramática) é por definição uma forma sentencial;

2. Seja $\alpha\rho\beta$ uma forma sentencial, com α e β cadeias quaisquer de terminais e/ou não-terminais da gramática, e seja $\rho \to \gamma$ uma produção da gramática. Nessas condições, a aplicação dessa produção à forma sentencial, substituindo a ocorrência de ρ por γ, produz uma nova forma sentencial $\alpha\gamma\beta$.

Denota-se a substituição acima definida, também conhecida como **derivação direta**, por:

$$\alpha\rho\beta \Rightarrow_G \alpha\gamma\beta$$

O índice "G" designa o fato de que a produção aplicada, no caso $\rho \to \gamma$, pertence ao conjunto de produções que define a gramática G. Nos casos em que a gramática em questão puder ser facilmente identificada, admite-se a eliminação de referências explícitas a ela. Note-se a distinção gráfica e de significado que se faz entre o símbolo empregado na denotação das produções da gramática ($\to$) e o símbolo utilizado na denotação das derivações ($\Rightarrow$).

Uma seqüência de zero ou mais derivações diretas, como, por exemplo, $\alpha \Rightarrow \beta \Rightarrow ... \Rightarrow \mu$ é chamada simplesmente **derivação**, e pode ser abreviada como $\alpha \Rightarrow^* \mu$.

Derivações em que ocorre a aplicação de pelo menos uma produção são denominadas **derivações não-triviais**, e são denotadas por $\alpha \Rightarrow^+ \mu$.

Se, pela aplicação de uma derivação não-trivial à raiz S de uma gramática, for possível obter uma cadeia w formada exclusivamente de símbolos terminais, diz-se que w, além de ser uma forma sentencial, é também uma **sentença**, e denota-se a sua derivação por:

$$S \Rightarrow^+ w$$

Gramáticas são sistemas formais baseados na substituição, em uma forma sentencial, de uma cadeia, coincidente com o lado esquerdo de uma regra de produção, pela cadeia correspondente ao lado direito da mesma regra, visando com isso, ao final de uma seqüência de substituições, a obtenção de uma cadeia sobre Σ. O processo de substituição tem início sempre a partir da raiz S da gramática, e é finalizado assim que for obtida uma forma sentencial isenta de símbolos não-terminais, isto é, assim que se obtiver uma sentença.

Exemplo 2.22 Considere-se a gramática G_1, definida no Exemplo 2.21.

- S é por definição uma forma sentencial;
- $0S33$ é uma forma sentencial, pois $S \Rightarrow 0S33$;
- $S \Rightarrow 0S33$ é uma derivação direta;

- $00S3333$ e $00A3333$ são formas sentenciais, pois $0S33 \Rightarrow 00S3333 \Rightarrow 00A3333$ através das produções $S \rightarrow 0S33$ e $S \rightarrow A$, aplicadas nesta ordem;
- $S \Rightarrow^+ 00A3333$ e $S \Rightarrow^+ 0S33$ são exemplos de derivações não-triviais;
- $00S3333 \Rightarrow^* 00S3333$ e $0S33 \Rightarrow 00A3333$ são exemplos de derivações;
- 12 e 00123333 são exemplos de sentenças, pois ambas são formadas exclusivamente por símbolos terminais e $S \Rightarrow A \Rightarrow 12$, ou seja, $S \Rightarrow^+ 12$, e $S \Rightarrow 0S33 \Rightarrow 00S3333 \Rightarrow 00A3333 \Rightarrow 00123333$, ou seja, $S \Rightarrow^+ 00123333$.

□

Ao conjunto de todas as sentenças w geradas por uma gramática G dá-se o nome de **linguagem definida pela gramática** G, ou simplesmente $L(G)$. Formalmente,

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^+ w\}$$

Exemplo 2.23 Pela inspeção das produções da gramática G_1 do Exemplo 2.21, pode-se concluir que:

$$L_1(G_1) = \{0^m 1^n 2^n 3^{2m} \mid m \geq 0 \text{ e } (n = 0 \text{ ou } n = 1)\}$$

São exemplos de sentenças pertencentes a L_1 : $\epsilon, 12, 033, 01233, 003333, 00123333$ etc. □

Como se pode perceber, diferentes seqüências de produções aplicadas à (única) raiz da gramática possibilitam a geração das diferentes (em geral, infinitas) sentenças da linguagem. Por outro lado, muitas vezes há mais de uma alternativa de substituição para um mesmo trecho de uma forma sentencial, ou, ainda, pode haver mais de um trecho em uma mesma forma sentencial que pode ser objeto de substituição pelo lado direito de alguma produção. Assim, a identificação correta das sentenças de uma dada linguagem deve ser feita levando-se em conta a possibilidade de ocorrência de todos esses fatores durante o processo de síntese de cadeias.

A completa identificação da linguagem gerada por uma determinada gramática é uma tarefa que exige abstração e alguma prática na manipulação das produções. Algumas gramáticas, como é o caso de G_1, podem ser analisadas sem dificuldade. Contudo, na prática, podem ocorrer gramáticas consideravelmente mais complexas. Observe-se, a título de ilustração, o caso da gramática G_2 apresentada no Exemplo 2.24.

Exemplo 2.24 Considere $G_2 = (V_2, \Sigma_2, P_2, S)$, com:

$$\begin{aligned} V_2 &= \{a, b, c, S, B, C\} \\ \Sigma_2 &= \{a, b, c\} \\ P_2 &= \{S \rightarrow aSBC, S \rightarrow abC, CB \rightarrow BC, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\} \end{aligned}$$

Uma análise mais profunda de G_2 permite concluir que $L_2(G_2) = \{a^n b^n c^n \mid n \geq 1\}$. A sentença $aabbcc$, por exemplo, é derivada da seguinte forma nessa gramática:

$$S \Rightarrow aSBC \Rightarrow aabCBC \Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbbcC \Rightarrow aabbcc$$

pela aplicação, respectivamente, das produções:

$$S \rightarrow aSBC, S \rightarrow abC, CB \rightarrow BC, bB \rightarrow bb, bC \rightarrow bc \text{ e } cC \rightarrow cc$$

□

É possível definir uma mesma linguagem através de duas ou mais gramáticas distintas. Quando isso ocorre, diz-se que as gramáticas que definem a linguagem em questão são sintaticamente equivalentes ou, simplesmente, **equivalentes** uma à outra.

Exemplo 2.25 As gramáticas G_3 e G_4 a seguir definidas são equivalentes:

$$\begin{aligned} G_3 &= (\{a,b,S\},\{a,b\},\{S \to aS, S \to a, S \to bS, S \to b, S \to aSb\}, S) \\ G_4 &= (\{a,b,S,X\},\{a,b\},\{S \to XS, S \to X, X \to a, X \to b\}, S) \end{aligned}$$

Uma rápida análise de G_3 e G_4 permite concluir que $L_3(G_3) = L_4(G_4) = \{a,b\}^+$. □

São muitas as notações (metalinguagens) utilizadas na definição gramatical das linguagens formais e de programação. Nos exemplos acima, bem como na maior parte do presente texto, utiliza-se a **notação algébrica**. No entanto, diversas outras metalinguagens são largamente empregadas para representar dispositivos generativos. A escolha de uma ou outra metalinguagem varia de acordo com o tipo da linguagem que estiver sendo definida, com o uso que se pretenda fazer de tal representação formal e com as próprias preferências pessoais de quem a estiver elaborando.

À exceção da notação algébrica, empregada para representar diversos tipos de linguagens e utilizada com especial ênfase no estudo teórico das propriedades das linguagens formais, outras notações, como, por exemplo, as Expressões Regulares, o BNF (*Backus-Naur Form* ou Notação de Backus-Naur), a Notação de Wirth (ou Expressões Regulares Estendidas) e os Diagramas de Sintaxe (também conhecidos como Diagramas Ferroviários), são bastante populares e amplamente utilizadas na definição de linguagens de programação de alto nível.

2.4 Linguagens, Gramáticas e Conjuntos

Linguagens são conjuntos. Conjuntos que formam linguagens são coleções de cadeias construídas sobre um alfabeto, por exemplo, através de gramáticas. A fim de explicitar e facilitar o entendimento de linguagens vistas como conjuntos, é conveniente considerar-se uma coleção finita de linguagens distintas, todas infinitas, definidas sobre um mesmo alfabeto qualquer. Analisando-se as respectivas gramáticas, sugere-se que o leitor procure compreender de que modo as linguagens foram definidas e, principalmente, verifique a relação que existe entre elas.

É interessante verificar: estas linguagens possuem elementos (sentenças) em comum? Qual é a sua intersecção? Alguma dessas linguagens é subconjunto de outra? Superconjunto? Subconjunto próprio? São inúmeras as possibilidades. O importante é desenvolver a percepção de que linguagens são conjuntos. Isso facilitará o estudo de novas operações sobre linguagens, e também de suas propriedades.

Exemplo 2.26 A gramática $G_0 = (\{a,b,S\},\{a,b\},\{S \to aS, S \to bS, S \to \epsilon\}, S)$ é tal que $L_0(G_0) = \Sigma^*$.

Substituindo a regra $S \to \epsilon$ pelas regras $S \to a$ e $S \to b$, em G, obtém-se uma nova gramática $G_1 = (\{a,b,S\},\{a,b\},\{S \to aS, S \to bS, S \to a, S \to b\}, S)$, de tal forma que agora $L_1(G_1) = \Sigma^+$.

A linguagem L_2, formada por cadeias sobre $\Sigma = \{a,b\}$, de tal modo que todas elas sejam iniciadas pelo símbolo a, é gerada pela gramática $G_2 = (\{a,b,S,X\},\{a,b\},\{S \to aX, X \to aX, X \to bX, X \to \epsilon\}, S)$. São exemplos de cadeias pertencentes a L_2 : a, abb, $abaaa$, $aabbbba$, aaa etc.

Por outro lado, considere-se a linguagem L_3, composta por cadeias sobre $\Sigma = \{a,b\}$, de tal forma que todas elas sejam iniciadas com o símbolo a e terminadas com o símbolo b. Uma possível gramática que gera L_3 é $G_3 = (\{a,b,S,X\},\{a,b\},\{S \to aX, X \to aX, X \to bX, X \to b\}, S)$. Perceba a sutil diferença que existe entre G_2 e G_3.

Em seguida, considere-se a linguagem L_4 que compreende todas as cadeias sobre $\Sigma = \{a,b\}$ que possuam exatamente dois símbolos b (nem mais, nem menos). São exemplos: bb, bab, abb, $aaaaabaab$ etc. L_4 é gerada pela gramática $G_4 = (\{a,b,S,X\},\{a,b\},\{S \to XbXbX, X \to aX, X \to \epsilon\}, S)$.

Finalmente, a linguagem L_5 é definida pelas cadeias sobre $\Sigma = \{a, b\}$, de tal forma que todas elas sejam iniciadas com o símbolo b e contenham um único símbolo b. São exemplos b, ba, baa, $baaa$ etc. A linguagem L_5 é gerada por $G_5 = (\{a, b, S, X\}, \{a, b\}, \{S \rightarrow bX, X \rightarrow aX, X \rightarrow \epsilon\}, S)$.

Esquematicamente, a relação entre as linguagens L_0, L_1, L_2, L_3, L_4 e L_5 pode ser observada na Figura 2.5 (é bom ter em mente que linguagens são conjuntos).

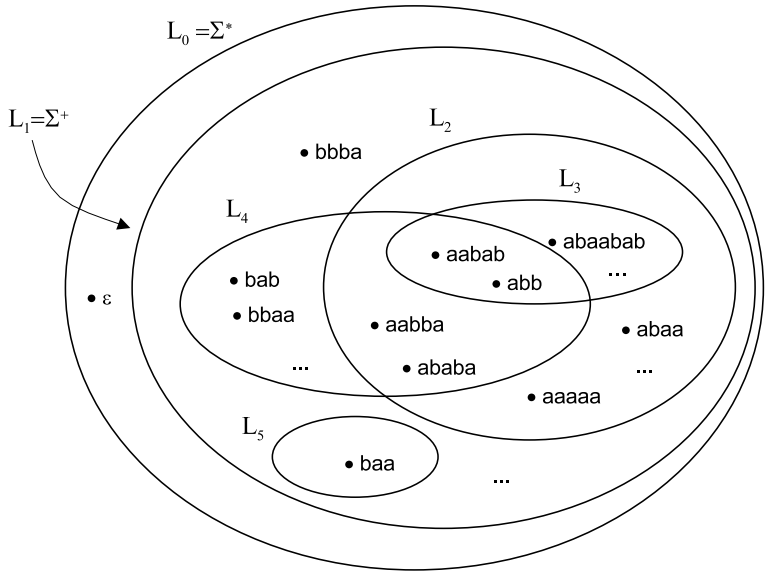

Figura 2.5: As linguagens do Exemplo 2.26 como conjuntos

Observe-se a relação de inclusão própria entre as linguagens estudadas. A cadeia ϵ ilustra a inclusão própria de L_1 em L. A cadeia $bbba$, de L_2 em L_1. Através da cadeia $abaa$, exemplifica-se a inclusão própria de L_3 em L_2. Sobre L_4 pode-se apenas dizer que está incluída propriamente em L_1.

A cadeia abb pertence simultaneamente às linguagens L_0, L_1, L_2, L_3 e L_4. A cadeia $abaabab$ pertence a L_0, L_1, L_2 e L_3 apenas. A cadeia $ababa$, somente às linguagens L_0, L_1, L_2 e L_4, e assim por diante.

Cumpre, novamente, notar que:

- A **maior linguagem** que pode ser definida sobre $\{a, b\}$ é $\{a, b\}^*$, que, neste caso, corresponde a L_0;
- O conjunto de **todas as linguagens** que podem ser definidas sobre $\{a, b\}$ é dado por $2^{\{a,b\}^*}$. Assim, L_0, L_1, L_2, L_3, L_4 e L_5 pertencem, todas, a $2^{\{a,b\}^*}$. Em outras palavras, cada uma dessas linguagens é um elemento de $2^{\{a,b\}^*}$, que por sua vez é um conjunto infinito.

□

2.5 Reconhecedores

Conhecidos também como **dispositivos cognitivos**, **dispositivos de aceitação**, **aceitadores sintáticos** ou simplesmente **autômatos**, os **reconhecedores** são sistemas formais capazes de aceitar todas as sentenças que pertençam a uma determinada linguagem, rejeitando todas as demais. Por esse motivo, constituem uma forma alternativa às gramáticas para a representação finita de linguagens.

Diferentemente das gramáticas, para as quais é possível definir e empregar uma notação formal adequada para todas as classes de linguagens (notação algébrica vista na Seção 2.3), os reconhecedores, ao contrário, são mais convenientemente estudados

se forem utilizadas notações (metalinguagens) diversas, cada qual apropriada para a correspondente classe de linguagens, de acordo com a classificação apresentada adiante, na Seção 2.6.

Não obstante, é possível estudar genericamente os reconhecedores do ponto de vista conceitual, sem entrar nos pormenores de sua formalização, uma vez que todos eles constituem instâncias de um modelo conceitual único. Assim, no presente texto, os aspectos de notação são considerados apenas no contexto do estudo de cada tipo de linguagem, sem prejuízo para o entendimento do modelo geral a seguir apresentado.

Os reconhecedores — esboçados em sua forma geral na Figura 2.6 — apresentam quatro componentes fundamentais: uma memória (fita) contendo o texto de entrada do reconhecedor, um cursor, que indica o próximo elemento da fita a ser processado, uma máquina de estados finitos, sem memória, e uma memória auxiliar opcional.

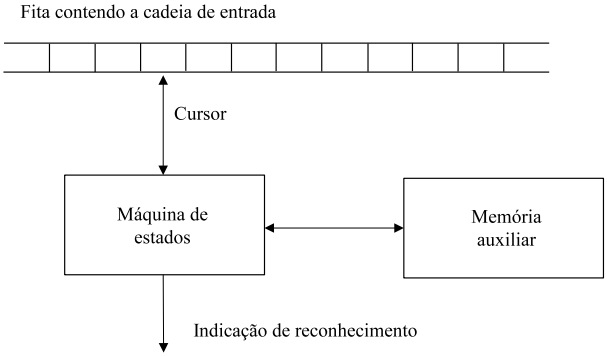

Figura 2.6: Organização de um reconhecedor genérico

A **fita de entrada** contém a cadeia a ser analisada pelo reconhecedor. Ela é dividida em células, e cada célula pode conter um único símbolo da cadeia de entrada, pertencente ao **alfabeto de entrada** escolhido para o reconhecedor. A cadeia de entrada é disposta da esquerda para a direita, sendo o seu primeiro símbolo colocado na posição mais à esquerda da fita.

Dependendo do tipo de reconhecedor considerado, a fita (ou o conjunto de fitas) de entrada pode apresentar comprimento finito ou infinito. Neste último caso, a fita pode ter ou não limitação à esquerda e/ou à direita. A cadeia de entrada registrada na fita de entrada pode estar delimitada por símbolos especiais, não pertencentes ao alfabeto de entrada, à sua esquerda e/ou à sua direita, porém isso não é obrigatório.

A leitura dos símbolos gravados na fita de entrada é feita através de um cabeçote de acesso, normalmente denominado **cursor**, o qual sempre aponta o próximo símbolo da cadeia a ser processado. Os movimentos do cursor são controlados pela máquina de estados, e podem, dependendo do tipo de reconhecedor, ser unidirecionais (podendo deslocar-se para um lado apenas, tipicamente para a direita) ou bidirecionais (podendo deslocar-se para a esquerda e para a direita). Determinados tipos de reconhecedores utilizam o cursor não apenas para lerem os símbolos da fita de entrada, mas também para escreverem sobre a fita, substituindo símbolos nela presentes por outros, de acordo com comandos determinados pela máquina de estados.

A **máquina de estados** funciona como um controlador central do reconhecedor, e contém uma coleção finita de **estados**, responsáveis pelo registro de informações colhidas

no passado, mas consideradas relevantes para decisões futuras, e **transições**, que promovem as mudanças de estado da máquina em sincronismo com as operações efetuadas através do cursor sobre a fita de entrada. Além disso, a máquina de estados finitos pode utilizar uma memória auxiliar para armazenar e consultar outras informações, também coletadas ao longo do processamento, que sejam eventualmente necessárias ao completo reconhecimento da cadeia de entrada.

A **memória auxiliar** é opcional, e torna-se necessária apenas em reconhecedores de linguagens que apresentam uma certa complexidade. Normalmente, ela assume a forma de uma estrutura de dados de baixa complexidade, como, por exemplo, uma pilha (no caso do reconhecimento de linguagens livres de contexto). As informações registradas na memória auxiliar são codificadas com base em um **alfabeto de memória**, e todas as operações de manipulação da memória auxiliar (leitura e escrita) fazem referência apenas aos símbolos que compõem esse alfabeto. Os elementos dessa memória são referenciados através de um **cursor auxiliar** que, eventualmente, poderá coincidir com o próprio cursor da fita de entrada. Seu tamanho não é obrigatoriamente limitado e, por definição, seu conteúdo pode ser consultado e modificado.

O diagrama abaixo resume os componentes de um reconhecedor genérico e as diversas formas como cada um deles pode se apresentar:

$$\text{Reconhecedor} \begin{cases} \text{Máquina de Estados} \begin{cases} \text{Finita} \end{cases} \\ \text{Fita de entrada + Cursor} \begin{cases} \text{Limitada / Não limitada} \\ \text{Leitura apenas / Leitura e escrita} \\ \text{Direita apenas / Direita e esquerda} \end{cases} \\ \text{Memória auxiliar + Cursor} \begin{cases} \text{Não limitada} \\ \text{Leitura e escrita} \end{cases} \end{cases}$$

A operação de um reconhecedor baseia-se em uma seqüência de movimentos que o conduzem, de uma configuração inicial única, para alguma configuração de parada, indicativa do sucesso ou do fracasso da tentativa de reconhecimento da cadeia de entrada.

Cada **configuração** de um autômato é caracterizada pela quádrupla:

1. Estado;
2. Conteúdo da fita de entrada;
3. Posição do cursor;
4. Conteúdo da memória auxiliar.

A **configuração inicial** de um autômato é definida como sendo aquela em que as seguintes condições são verificadas:

1. Estado: inicial, único para cada reconhecedor;
2. Conteúdo da fita de entrada: com a cadeia completa a ser analisada;
3. Posição do cursor: apontando para o símbolo mais à esquerda da cadeia;
4. Conteúdo da memória auxiliar: inicial, pré-definido e único.

A **configuração final** de um autômato é aquela na qual as seguintes condições são obedecidas:

1. Estado: algum dos estados finais, que não são necessariamente únicos no reconhecedor;

2. Conteúdo da fita de entrada: inalterado ou alterado, em relação à configuração inicial, conforme o tipo de reconhecedor;

3. Posição do cursor: apontando para a direita do último símbolo da cadeia de entrada ou apontando para qualquer posição da fita, conforme o tipo de reconhecedor;

4. Conteúdo da memória auxiliar: final e pré-definido, não necessariamente único ou idêntico ao da configuração inicial, ou apenas indefinido.

A especificação de uma possibilidade de **movimentação** entre uma configuração e outra é denominada **transição**. A movimentação do autômato da configuração corrente para uma configuração seguinte é feita, portanto, levando-se em conta todas as transições passíveis de serem aplicadas pelo reconhecedor à configuração corrente. Uma transição mapeia triplas formadas por:

- Estado corrente;
- Símbolo correntemente apontado pelo cursor da fita de entrada;
- Símbolo correntemente apontado pelo cursor da memória auxiliar;

em triplas formadas por:

- Próximo estado;
- Símbolo que substituirá o símbolo correntemente apontado pelo cursor da fita de entrada e o sentido do deslocamento do cursor;
- Símbolo que substituirá o símbolo correntemente apontado pelo cursor da memória auxiliar.

A identificação das transições passíveis de serem aplicadas a uma determinada configuração é determinada pela máquina de estados. Ela também é responsável, na ocasião da aplicação de uma transição, pelo deslocamento do cursor, acompanhado de uma eventual substituição do símbolo lido na fita de entrada, e também pela eventual modificação do conteúdo da memória auxiliar.

Não se devem confundir os conceitos de movimentação e de transição, uma vez que o simples fato de o autômato exibir várias transições que sejam aplicáveis a uma dada configuração não implica que haja obrigatoriamente mais de uma possibilidade de movimentação em tal configuração.

Essa multiplicidade de possibilidades ocorrerá apenas se houver mais de uma transição, associada ao mesmo estado, que faça referência ao mesmo símbolo de entrada e ao mesmo conteúdo da memória auxiliar. **Transições em vazio**, ou seja, transições que permitem a movimentação do autômato sem levar em conta os símbolos da cadeia

de entrada, também constituem possibilidades adicionais de movimentação se estiverem associadas a outras transições (vazias ou não-vazias) em uma mesma configuração.

Diz-se que um autômato **aceita** (ou reconhece) uma cadeia se lhe for possível atingir alguma configuração final a partir de sua configuração inicial única, através de movimentos executados sobre tal cadeia. Caso contrário, diz-se que o autômato **rejeita** a cadeia. A maneira como tais configurações sucedem umas às outras durante o reconhecimento (ou aceitação) da cadeia de entrada define uma característica fundamental dos autômatos, conforme explicado a seguir.

É possível que ocorram, durante o reconhecimento de uma cadeia de entrada, configurações para as quais se apresentem nenhuma, apenas uma ou eventualmente diversas possibilidades distintas de movimentação.

No caso de haver uma única possibilidade de movimentação, diz-se que a transição efetuada é determinística, dado que não existem outras opções de evolução para o autômato em tal configuração. Quando todos os movimentos passíveis de serem executados por um autômato forem movimentos determinísticos, independentemente de qual seja a cadeia de entrada, e só neste caso, diz-se que o autômato é **determinístico**.

No caso de haver mais de uma alternativa de movimentação em uma ou mais configurações do autômato (independentemente da cadeia de entrada), todas as possibilidades devem ser consideradas legítimas, e o autômato deverá optar, sucessiva e aleatoriamente, por todas as alternativas apresentadas, uma de cada vez, até que uma das duas seguintes condições seja satisfeita:

1. Uma configuração final seja atingida, ou

2. Esgotadas todas as possibilidades, nenhuma configuração final possa ser atingida.

Um autômato com tal característica é denominado **não-determinístico**. Uma cadeia de entrada é aceita por ele se pelo menos uma das múltiplas seqüências de movimentações realizadas pelo autômato for capaz de conduzi-lo a alguma configuração final, partindo de sua configuração inicial única. Diz-se que o autômato rejeita a cadeia de entrada quando, esgotadas todas as seqüências de movimentação possíveis, o mesmo não puder atingir uma configuração final qualquer.

Do ponto de vista conceitual, o não-determinismo de um autômato pode ser entendido a partir do modelo de paralelismo que é inerente à sua operação. De acordo com esse modelo, sempre que o autômato se depara com mais de uma possibilidade de movimentação, considera-se que múltiplas instâncias do mesmo são criadas (em número igual à quantidade de movimentações distintas previstas pela função de transição, a partir da configuração corrente), cada qual herdando da instância anterior a configuração corrente do autômato (estado, situação da cadeia de entrada e situação da memória auxiliar), porém evoluindo para configurações distintas. Dessa forma, todas as possibilidades são consideradas.

Fazendo-se uma analogia com o modelo de implementação baseado em programação, o modelo paralelo de operação corresponde à criação de múltiplos "threads" de execução, cada qual sendo responsável pela verificação de uma seqüência possível de movimentações. Autômatos determinísticos, por um lado, podem ser facilmente abstraídos a partir do modelo seqüencial, não-paralelo de computação, uma vez que não existe a necessidade de se considerarem vários "threads". Por outro lado, como se pode intuir, a criação de modelos de execução para autômatos não-determinísticos em máquinas estritamente seqüenciais não é direta, requerendo do programador e/ou do ambiente de execução artifícios que viabilizem o modelo paralelo de computação.

Quando não há alternativas de movimentação a partir da configuração corrente do autômato, constata-se um impasse, caracterizando a impossibilidade de prosseguimento da operação de reconhecimento. Se o autômato for determinístico, isso será interpretado como sucesso do reconhecimento se e apenas se for atingida alguma configuração de aceitação (configuração final), sendo constatado um fracasso na operação de reconhecimento nos demais casos.

Caso o autômato seja do tipo não-determinístico, tal impasse indicará apenas que a particular seqüência de movimentações em andamento ("thread") não conduziu ao reconhecimento da sentença. Diz-se que a cadeia é aceita por um autômato não-determinístico se pelo menos um "thread" for capaz de conduzi-lo até impasse em uma configuração final. Caso todas as seqüências ("threads") conduzam a impasses em configurações não-finais, diz-se que o autômato rejeita a cadeia de entrada.

Seja o autômato determinístico ou não-determinístico, a linguagem por ele aceita (ou definida) corresponde ao conjunto de todas as cadeias que ele aceita.

Do ponto de vista prático, existe um interesse muito maior nos autômatos determinísticos, visto que sua operação não exige repetidas pesquisas exaustivas em árvores de alternativas. Tal fato possibilita a construção de autômatos eficientes, constituindo uma opção muito atraente para várias aplicações, como é o caso do projeto da lógica de controladores finitos, ou de compiladores para linguagens de programação de alto nível.

Os conceitos acima apresentados podem ser formalizados representando-se os reconhecedores como dispositivos cujas configurações são definidas como elementos do produto cartesiano:

$$Q \times (\Sigma^* \times \Sigma^*) \times \Gamma^*$$

onde:

- Q é o conjunto de estados do controle finito;
- Σ é o alfabeto de entrada do reconhecedor;
- Γ é o alfabeto de sua memória auxiliar.

Dessa forma, as **configurações** de um reconhecedor genérico podem ser representadas como triplas $(q, (\alpha, \beta), \gamma)$, onde $q \in Q, (\alpha, \beta) \in \Sigma^* \times \Sigma^*$ e $\gamma \in \Gamma^*$.

Para cada configuração assim representada, o termo q designa o estado específico assumido pela máquina de estados, o termo γ representa o conteúdo completo da memória auxiliar, e o par ordenado (α, β) indica ao mesmo tempo o conteúdo completo da fita de entrada e a posição do cursor sobre a mesma, da seguinte forma:

- α representa o conteúdo da fita à **esquerda** do cursor;
- β representa o conteúdo da fita à **direita** do cursor, incluindo o símbolo apontado pelo cursor, na configuração considerada.

Interpreta-se α como a parte da fita já analisada pelo reconhecedor, e β a parte ainda a analisar.

Assim, por exemplo, se $\beta = \sigma\,\mu$, com $\sigma \in \Sigma$ e $\mu \in \Sigma^*$, então σ é o símbolo apontado pelo cursor nessa configuração, e μ corresponde à cadeia situada à sua direita.

As transições de um reconhecedor podem ser representadas como funções totais δ que mapeiam configurações correntes em novas configurações:

$$\delta : Q \times (\Sigma^* \times \Sigma^*) \times \Gamma^* \rightarrow 2^{Q \times (\Sigma^* \times \Sigma^*) \times \Gamma^*}$$

Como normalmente o cursor permite o acesso a apenas um símbolo da fita de entrada de cada vez (aquele apontado pelo cursor em cada configuração), a função δ costuma ser simplificada para:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^* \rightarrow 2^{Q \times (\Sigma^* \times \Sigma^*) \times \Gamma^*}$$

Além disso, é usual que a consulta ao conteúdo da memória auxiliar seja feita levando-se em consideração apenas um símbolo do alfabeto Γ de cada vez (o critério de escolha do símbolo a ser consultado depende do tipo de autômato e da forma como a memória auxiliar está organizada). Assim, a função δ sofre uma nova simplificação:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times (\Sigma^* \times \Sigma^*) \times \Gamma^*}$$

Ainda por uma questão de praticidade, em vez de se especificar os elementos de $\Sigma^* \times \Sigma^*$ e Γ^* que deverão substituir integralmente a cadeia de entrada e o conteúdo da memória auxiliar após a execução da respectiva transição pelo autômato, costuma-se especificar apenas o efeito que tal transição produz sobre a configuração corrente.

No caso da cadeia de entrada, especifica-se apenas o novo símbolo que substituirá o símbolo recém-lido, bem como o sentido de deslocamento do cursor (para a esquerda ou para a direita) após ser efetuada essa substituição. No caso da memória auxiliar, especifica-se uma cadeia, que deverá substituir o símbolo consultado pela transição executada. Para tanto, a função δ assume a sua nova e definitiva forma:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times ((\Sigma \cup \{\epsilon\}) \times \{E,D,\epsilon\}) \times \Gamma^*}$$

$\{E, D, \epsilon\}$ é um conjunto cujos elementos indicam as possibilidades de sentido de deslocamento do cursor após a consulta do símbolo corrente na cadeia de entrada pelo autômato. E indica deslocamento para a esquerda, D para a direita e ϵ ausência de movimento do cursor.

A **função de transição** especifica um conjunto de possibilidades de movimentação, podendo, em cada uma, produzir os seguintes efeitos sobre a configuração corrente do autômato:

1. Mudança no estado corrente de q_i para q_j com $q_i, q_j \in Q$;

2. Alteração da cadeia de entrada a partir da consulta do símbolo corrente $\sigma_1 \in \Sigma$ e de sua subseqüente substituição por um novo elemento $\sigma_2 \in \Sigma$;

3. Eventual modificação da cadeia de entrada sem que haja consulta do símbolo corrente (caso $\{\epsilon\}$ no domínio de δ); por outro lado, é também possível que não haja modificação da cadeia de entrada como resultado da aplicação da transição (caso $\{\epsilon\}$ no contradomínio de δ);

4. Eventual deslocamento do cursor da fita de entrada para a direita ou para a esquerda, após a aplicação da transição; no caso ϵ em $\{E, D, \epsilon\}$, o cursor permanece na posição original, sem se deslocar;

5. Alteração do conteúdo da memória auxiliar, a partir da consulta de um símbolo $\gamma_1 \in \Gamma$ e de sua subseqüente substituição por uma cadeia $\gamma_2 \in \Gamma^*$;

6. Deslocamento do cursor da memória auxiliar, de forma compatível com a maneira como a mesma estiver estruturada. Exemplo: tipicamente, para memória auxiliar

estruturada como pilha, o cursor se move de forma compatível com as operações de movimentação das pilhas, ou seja, realizando operações de "*push*" e "*pop*".

Note, adicionalmente, que:

7. É possível associar uma mesma configuração corrente a mais de uma nova configuração seguinte (uma possível forma de não-determinismo);

8. É possível haver configurações χ tais que $\delta(\chi) = \varnothing$. Trata-se de configurações para as quais não foram definidas possibilidades de movimentação (nessas configurações o reconhecedor pára por falta de opções para prosseguir as operações de reconhecimento).

A operação de movimentação de um reconhecedor, de uma dada configuração para a configuração seguinte, por meio da aplicação de uma transição, é denotada através do símbolo "$\vdash$":

$$\vdash: Q \times (\Sigma^* \times \Sigma^*) \times \Gamma^* \rightarrow Q \times (\Sigma^* \times \Sigma^*) \times \Gamma^*$$

Denota-se uma determinada movimentação do reconhecedor como:

$$(q_i, (\alpha, \beta), \gamma) \vdash (q_j, (\phi, \nu), \eta)$$

O símbolo "$\vdash$" denota, portanto, uma relação binária sobre o conjunto das configurações $Q \times (\Sigma^* \times \Sigma^*) \times \Gamma^*$ de cada autômato.

Exemplo 2.27 Considere-se o seguinte autômato, parcialmente definido:

$$Q = \{q_0, q_1\};\ \Sigma = \{0, 1\};\ \Gamma = \{A, B\},\ \text{com:}$$

1. $\delta(q_0, 0, A) = \{(q_1, (0, D), AA)\}$
2. $\delta(q_0, 0, B) = \{(q_1, (0, D), \epsilon)\}$
3. $\delta(q_1, \epsilon, A) = \{(q_1, (0, D), BA)\}$
4. $\delta(q_1, 1, B) = \{(q_1, (1, E), AB)\}$
5. $\delta(q_1, 0, A) = \{(q_0, (0, E), B)\}$

Supondo que a cadeia de entrada seja 0110, que o conteúdo corrente da memória auxiliar seja A e que o cursor de leitura esteja apontando para o símbolo mais à esquerda da cadeia de entrada, a seguinte seqüência de movimentos poderia fazer parte do reconhecimento de tal cadeia:

$$\begin{aligned}
& (q_0, (\epsilon, 0110), A) \\
\vdash\ & (q_1, (0, 110), AA) \\
\vdash\ & (q_1, (00, 10), BAA) \\
\vdash\ & (q_1, (0, 010), ABAA) \\
\vdash\ & (q_0, (\epsilon, 0010), BBAA) \\
\vdash\ & (q_1, (0, 010), BAA)
\end{aligned}$$

Tais movimentos correspondem, respectivamente, à aplicação das transições 1, 3, 4, 5 e 2, nesta ordem.

Naturalmente, outros movimentos poderiam ter sido executados partindo-se da mesma configuração inicial. Por exemplo, a transição 5 poderia ter sido executada na configuração em que se faz uso da transição 3 e vice-versa. Neste caso, uma nova seqüência de movimentações seria obtida, correspondendo à aplicação das transições 1, 5, 2, 3 e 4, nesta ordem:

$$
\begin{aligned}
& (q_0, (\epsilon, 0110), A) \\
\vdash\ & (q_1, (0, 110), AA) \\
\vdash\ & (q_0, (\epsilon, 0110), BA) \\
\vdash\ & (q_1, (0, 110), A) \\
\vdash\ & (q_1, (00, 10), BA) \\
\vdash\ & (q_1, (0, 010), ABA)
\end{aligned}
$$

□

Um reconhecedor é considerado não-determinístico se e apenas se pelo menos uma das condições abaixo, referentes à definição da função δ, for verificada. Caso contrário, ele será determinístico:

- $\exists\, q \in Q, \sigma \in \Sigma, \gamma \in \Gamma$ tal que $\left|\delta(q, \sigma, \gamma)\right| \geq 2$;
- $\exists\, q \in Q, \gamma \in \Gamma$ tal que $\left|\delta(q, \epsilon, \gamma)\right| \geq 2$;
- $\exists\, q \in Q, \sigma \in \Sigma, \gamma \in \Gamma$ tal que, simultaneamente, $\left|\delta(q, \sigma, \gamma)\right| \geq 1$ e $\left|\delta(q, \epsilon, \gamma)\right| \geq 1$.

A verificação de qualquer das condições acima implica a necessidade de o reconhecedor optar, a partir de uma dada configuração corrente, por uma dentre uma coleção de novas configurações possíveis, caracterizando dessa maneira o não-determinismo de sua operação.

No primeiro caso, o não-determinismo é identificado pela existência de mais de uma possibilidade de movimentação a partir de uma dada configuração corrente, possibilidades estas que são expressas em termos das indicações de um estado corrente, de um símbolo do alfabeto de entrada e de um símbolo da memória auxiliar; no segundo caso, pela existência de mais de uma possibilidade de movimentação a partir de uma dada configuração corrente, possibilidades estas expressas apenas em função do estado corrente e do símbolo do alfabeto da memória auxiliar (sem referências ao alfabeto de entrada); finalmente, no terceiro caso, pela coexistência, em um dada configuração corrente, de possibilidades de movimentação com ou sem referências a símbolos do alfabeto de entrada (pelo menos uma de cada tipo).

Note-se, portanto, que uma transição (ou especificação de uma possibilidade de movimentação) possui caráter estático, ou seja, ela está associada a uma descrição estática do comportamento de um reconhecedor. Transições descrevem movimentações potenciais, as quais só se concretizam, durante a operação do reconhecedor, ao serem habilitadas pelas configurações assumidas pelo mesmo.

Por outro lado, as movimentações possuem um caráter dinâmico e são conseqüências diretas da aplicação de transições sobre as configurações sucessivamente assumidas pelo reconhecedor. Observado por esse prisma, um reconhecedor pode ser considerado não-determinístico se exibir pelo menos uma configuração que possa evoluir, conforme as especificações de suas transições, para mais de uma possível configuração. Caso contrário, ele será dito determinístico.

Exemplo 2.28 Considere-se um reconhecedor cuja configuração corrente seja:

$$(q_1, (\alpha, \sigma_1\, \mu), \gamma_1\gamma), \quad q_1 \in Q, \sigma_1 \in \Sigma, \alpha, \mu \in \Sigma^*, \gamma_1 \in \Gamma, \gamma \in \Gamma^*$$

σ_1 e γ_1 correspondem, respectivamente, aos símbolos correntemente referenciados pelos cursores da fita de entrada e da memória auxiliar. Seja δ uma função que especifique, entre outras, a seguinte transição:

$$\delta(q_1, \sigma_1, \gamma_1) = \{(q_2, (\sigma_2, D), \gamma_2)\}, \quad \sigma_2 \in \Sigma^*, \gamma_2 \in \Gamma^*$$

Suponha-se, ainda, que esta seja a única transição aplicável à configuração corrente. Neste caso, a movimentação do reconhecedor da configuração corrente para a seguinte, pela aplicação dessa transição, é determinística e representada por:

1. $(q_1, (\alpha, \sigma_1\, \mu), \gamma_1\gamma) \vdash (q_2, (\alpha\sigma_2, \mu), \gamma_2\gamma)$

Suponha-se, agora, a função de transição modificada como segue:

$$\delta(q_1, \sigma_1, \gamma_1) = \{(q_2, (\sigma_2, D), \gamma_2), (q_3, (\sigma_3, D), \gamma_3)\}$$

Nesta situação, haveria não apenas uma, mas duas movimentações possíveis a partir da configuração corrente:

1. $(q_1, (\alpha, \sigma_1\, \mu), \gamma_1\gamma) \vdash (q_2, (\alpha\sigma_2, \mu), \gamma_2\gamma)$, ou
2. $(q_1, (\alpha, \sigma_1\, \mu), \gamma_1\gamma) \vdash (q_3, (\alpha\sigma_3, \mu), \gamma_3\gamma)$

Isso basta, neste exemplo, para constatar o caráter não-determinístico desse reconhecedor.

Finalmente, suponha-se que a função δ seja novamente modificada, incorporando, em adição à definição anterior, o elemento:

$$\delta(q_1, \epsilon, \gamma_1) = \{(q_4, (\epsilon, \epsilon), \gamma_4)\}$$

Isso caracteriza a coexistência de uma transição em vazio com duas transições não-vazias, todas aplicáveis a uma mesma configuração. Como conseqüência, a quantidade de alternativas de movimentação do reconhecedor, na configuração original, cresceria de duas para três:

1. $(q_1, (\alpha, \sigma_1\, \mu), \gamma_1\gamma) \vdash (q_2, (\alpha\sigma_2, \mu), \gamma_2\gamma)$
2. $(q_1, (\alpha, \sigma_1\, \mu), \gamma_1\gamma) \vdash (q_3, (\alpha\sigma_3, \mu), \gamma_3\gamma)$
3. $(q_1, (\alpha, \sigma_1\, \mu), \gamma_1\gamma) \vdash (q_4, (\alpha, \sigma_1\, \mu), \gamma_4\gamma)$

□

Reconhecedores determinísticos costumam ter suas transições definidas através de uma versão simplificada e particularizada da função δ genérica, visando com isso explicitar a associação unívoca entre cada uma de suas configurações com uma única configuração seguinte, conforme mostrado a seguir:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times (\Sigma \times \{E, D\}) \times \Gamma^*$$

Transições em vazio, em reconhecedores determinísticos, são admitidas apenas em configurações para as quais não existam outras possibilidades de movimentação, vazias ou não-vazias, visando, dessa maneira, preservar o caráter determinístico da escolha da transição a ser aplicada.

A **configuração inicial** de um reconhecedor costuma ser expressa como:

$$(q_0, (\epsilon, w), \gamma_0)$$

- $q_0 \in Q$ é o estado inicial;
- $w \in \Sigma^*$ é a cadeia de entrada a ser analisada;
- $\gamma_0 \in \Gamma^*$ é o conteúdo inicial da memória auxiliar;

Definindo-se $Q_F \subseteq Q$ como sendo o conjunto dos estados finais do reconhecedor, e $\Gamma_F \subseteq \Gamma^*$ como o conjunto dos conteúdos possíveis da memória auxiliar, convencionados como sendo finais, as **configurações finais** de um reconhecedor podem ser caracterizadas como:

$$(q_F, (\alpha, \beta), \gamma_F)$$

- $q_F \in Q_F$
- $\gamma_F \in \Gamma_F$
- Em particular, para os reconhecedores estudados nos Capítulos 3 e 4, $\alpha = w$ e $\beta = \epsilon$, ou seja, a cadeia de entrada deve necessariamente ser esgotada e a cadeia de entrada deve permanecer inalterada. Para os reconhecedores dos Capítulos 5, 6 e 7, α e β são quaisquer, indicando com isso que o cursor pode ser deixado em qualquer posição da fita e, também, que o conteúdo original da mesma não precisa ser necessariamente preservado ao término do processamento.

A linguagem L aceita por um reconhecedor M, definida como o conjunto de cadeias capaz de movimentá-lo desde a sua configuração inicial até alguma configuração final, pode, portanto, ser definida formalmente como:

$$L(M) = \{w \in \Sigma^* \mid (q_0, (\epsilon, w), \gamma_0) \vdash^* (q_F, (\alpha, \beta), \gamma_F)\}$$

Note-se que "$\vdash^*$" denota novamente o fechamento reflexivo e transitivo da relação "$\vdash$", ou seja, a movimentação do reconhecedor ao longo de zero ou mais configurações sucessivas.

O modelo genérico de reconhecedor apresentado nesta seção é adequado para as especializações que são discutidas nos capítulos seguintes, conforme a classe de linguagens que se esteja considerando. Tais reconhecedores especializados são utilizados no desenvolvimento teórico da matéria, incluindo a demonstração de teoremas e também de diversas propriedades importantes para cada uma dessas classes de linguagens.

Quando, no entanto, se consideram modelos de implementação para tais reconhecedores, em particular quando se trata de modelos de simulação por software, muitas vezes é conveniente acrescentar à sua especificação extensões que costumam facilitar a realização de tais modelos.

Entre as extensões mais comuns, é usual encontrar-se a inserção de delimitadores especiais em torno da cadeia de entrada (símbolos BOF e EOF, "begin-of-file" e "end-of-file", respectivamente indicativos de início e fim da cadeia de entrada), e também o desmembramento da caracterização da configuração do reconhecedor em termos da caracterização da configuração de cada uma das suas partes (da máquina de estados, da cadeia de entrada e da memória auxiliar).

Tais extensões não apresentam qualquer prejuízo ao modelo ou aos resultados teóricos apresentados, e constituem, como mencionado, mera conveniência de representação conforme a tecnologia de implementação adotada.

No caso particular da classe de reconhecedores estudados nos Capítulos 5, 6 e 7, o modelo teórico (Máquina de Turing) também faz uso dos delimitadores BOF e EOF. Nesses casos, a definição de configuração inicial estabelece que o cursor de acesso deve apontar inicialmente para o símbolo imediatamente à direita do BOF, símbolo esse que, por isso, corresponde ao segundo símbolo da cadeia de entrada (da esquerda para a direita) e ao primeiro símbolo a ser analisado de fato.

2.6 Hierarquia de Chomsky

O estudo sistemático das linguagens formais teve um forte impulso no final da década de 1950, quando o lingüista Noam Chomsky publicou dois artigos ([61] e [62]) apresentando o resultado de suas pesquisas relativas à classificação hierárquica das linguagens. Até então, a teoria dos autômatos se apresentava razoavelmente evoluída, porém a das linguagens formais ainda não se havia, de fato, estabelecido como disciplina.

A partir da publicação dos referidos artigos, houve uma significativa concentração de pesquisas na área das linguagens formais, e a teoria resultante, juntamente com a teoria dos autômatos, teve a oportunidade de se consolidar definitivamente, a partir do final da década de 1960, como uma disciplina coesa e fundamental para as áreas de engenharia e de ciência da computação.

Como teórico e estudioso das linguagens naturais, Chomsky se dedicava à pesquisa de modelos que permitissem a formalização de tais linguagens. Ainda hoje esse objetivo parece um tanto ambicioso, porém o trabalho de Chomsky logo chamou a atenção de especialistas de outras áreas, em particular os da área de computação, que viam, para suas teorias, grande aplicabilidade para a formalização e o estudo sistemático de linguagens artificiais, especialmente as de programação.

A prática confirmou a intuição dos especialistas. A classificação das linguagens, por ele proposta, conhecida como **Hierarquia de Chomsky**, tem como principal mérito agrupar as linguagens em classes, de tal forma que elas possam ser hierarquizadas de acordo com a sua complexidade relativa. Como resultado, é possível antecipar as propriedades fundamentais exibidas por uma determinada linguagem, ou mesmo vislumbrar os modelos de implementação mais adequados à sua realização, conforme a classe a que pertença.

Assim, o interesse prático pela Hierarquia de Chomsky se deve especialmente ao fato de ela viabilizar a escolha da forma mais econômica para a realização dos reconhecedores das linguagens, de acordo com a classe a que elas pertençam nessa hierarquia, evitando-se, portanto, o uso de formalismos mais complexos que o necessário, e o emprego de reconhecedores desnecessariamente ineficientes para as linguagens de menor complexidade. De um ponto de vista estritamente de engenharia, a Hierarquia de Chomsky permite determinar e selecionar o modelo de implementação (no que diz respeito apenas ao reconhecedor sintático) de menor custo para cada linguagem considerada.

A Hierarquia de Chomsky define quatro classes distintas de linguagens, denominadas tipos 0, 1, 2 e 3, as quais são geradas por gramáticas particularizadas em relação ao caso geral anteriormente apresentado, por intermédio de restrições que são aplicadas ao formato das produções $\alpha \rightarrow \beta$.

Dá-se o nome de **gramática linear à direita** àquela cujas produções obedeçam todas às seguintes condições:

- $\alpha \in N$
- $\beta \in \Sigma, \beta \in N, \beta \in \Sigma N$, ou $\beta = \epsilon$, de forma não exclusiva.

Exemplo 2.29 A gramática $G_1 = (\{0,1,2,3,S,A\}, \{0,1,2,3\}, \{S \rightarrow 0S, S \rightarrow 1S, S \rightarrow A, A \rightarrow 2, A \rightarrow 3\}, S)$ é linear à direita. □

Gramática linear à esquerda é aquela em que todas as produções exibem as seguintes características:

- $\alpha \in N$

- $\beta \in \Sigma, \beta \in N, \beta \in N\Sigma$, ou $\beta = \epsilon$, de forma não exclusiva.

Exemplo 2.30 A gramática $G_2 = (\{0, 1, 2, 3, S, A\}, \{0, 1, 2, 3\}, \{S \rightarrow S2, S \rightarrow S3, S \rightarrow A, A \rightarrow 1, A \rightarrow 0\}, S)$ é linear à esquerda. □

Gramáticas lineares, à esquerda ou à direita, também conhecidas como gramáticas do **tipo 3**, geram linguagens denominadas **regulares**, ou simplesmente do **tipo 3**. As linguagens regulares constituem a classe de linguagens mais simples dentro da Hierarquia de Chomsky, a qual prossegue com linguagens de complexidade crescente até as linguagens mais gerais, do tipo 0.

Uma gramática é dita **livre de contexto**, ou do **tipo 2**, se as suas produções possuírem apenas um símbolo não-terminal em seu lado esquerdo, e uma combinação qualquer de símbolos terminais e não-terminais no lado direito. Gramáticas desse tipo geram linguagens denominadas **livres de contexto**, ou do **tipo 2**. Formalmente:

- $\alpha \in N$
- $\beta \in V^*$

Exemplo 2.31 A gramática $G_3 = (\{0, 1, S\}, \{0, 1\}, \{S \rightarrow 0S1, S \rightarrow \epsilon\}, S)$ é livre de contexto. □

Deve-se notar que toda gramática do tipo 3 também se enquadra na definição de gramática do tipo 2, constituindo caso particular deste último. Logo, é correto dizer que toda gramática do tipo 3 é também uma gramática do tipo 2. Por outro lado, nem toda gramática do tipo 2 pode ser caracterizada também como gramática do tipo 3. Existe, portanto, uma relação de inclusão própria entre o conjunto das gramáticas do tipo 3 e o conjunto das gramáticas do tipo 2.

Exemplo 2.32 As gramáticas G_1 e G_2 são simultaneamente lineares e livres de contexto. A gramática G_3 é livre de contexto porém não é regular. □

Conseqüentemente, é correto dizer que todas as linguagens do tipo 3 (aquelas geradas por gramáticas do tipo 3) também são do tipo 2, uma vez que as gramáticas do tipo 3 são, por definição, do tipo 2. Na prática, no entanto, costuma-se designar o tipo de uma linguagem conforme o tipo da gramática mais simples que a gera. Conforme discutido nos capítulos seguintes, nem toda linguagem do tipo 2 pode ser caracterizada, inversamente, como sendo do tipo 3. Isso significa que existem linguagens do tipo 2 para as quais não é possível obter uma gramática do tipo 3. Logo, as linguagens do tipo 3 constituem um subconjunto próprio das linguagens do tipo 2.

As gramáticas do **tipo 1**, também conhecidas como **sensíveis ao contexto**, constituem a classe seguinte de gramáticas, na Hierarquia de Chomsky. A caracterização das gramáticas sensíveis ao contexto decorre da restrição, imposta ao formato das produções, de que o comprimento da cadeia do lado direito de cada produção seja no mínimo igual ao comprimento da cadeia do lado esquerdo, não havendo, portanto, possibilidade de redução do comprimento das formas sentenciais quando da realização de derivações em gramáticas deste tipo:

- $\alpha \in V^*NV^*$
- $\beta \in V^*$
- $|\beta| \geq |\alpha|$

Note-se que a definição de gramáticas sensíveis ao contexto não permite, *a priori*, que as respectivas linguagens por elas geradas incluam a cadeia vazia, justamente devido ao fato de que $|\beta| \geq |\alpha|$. No entanto, é comum (ver [49]) se considerar L uma linguagem **sensível ao contexto**, ou simplesmente do **tipo 1**, mesmo que $\epsilon \in L$, se $L - \{\epsilon\}$ puder ser gerada por uma gramática sensível ao contexto. Em outras palavras, linguagens sensíveis ao contexto são aquelas que são geradas por gramáticas sensíveis ao contexto, com a eventual incorporação da cadeia vazia.

Exemplo 2.33 A gramática $G_4 = (\{a, b, c, S, X, Y\}, \{a, b, c\}, \{S \rightarrow aXb, S \rightarrow aXa, Xa \rightarrow bc, Xb \rightarrow cb\}, S)$ é sensível ao contexto. □

Nem toda gramática do tipo 2 pode ser considerada uma gramática do tipo 1. De fato, gramáticas do tipo 2 permitem a geração da cadeia vazia, ao passo que gramáticas do tipo 1 não prevêem essa possibilidade. Nem toda gramática do tipo 1, no entanto, pode ser considerada uma gramática do tipo 2. Em particular, aquelas que apresentam produções cujo lado esquerdo seja composto por cadeias de dois ou mais símbolos.

Exemplo 2.34 As gramáticas lineares G_1 e G_2 são também sensíveis ao contexto. A gramática livre de contexto G_3, no entanto, não é sensível ao contexto, devido à presença da produção $S \rightarrow \epsilon$. □

Por outro lado, é possível provar, como veremos nos próximos capítulos, que toda linguagem livre de contexto é também uma linguagem sensível ao contexto. O contrário, porém, não é verdadeiro, pois existem linguagens sensíveis ao contexto, inclusive linguagens que não contêm a cadeia vazia, que não podem ser geradas por gramáticas livres de contexto, quaisquer que sejam estas. Logo, existe uma relação de inclusão própria entre as linguagens livres de contexto e as linguagens sensíveis ao contexto.

As gramáticas pertencentes à última classe definida pela Hierarquia de Chomsky recebem a denominação de **irrestritas**, ou do **tipo 0**. Como o próprio nome sugere, trata-se de gramáticas sobre as quais não é imposta nenhuma restrição quanto ao formato de suas produções, exceto pelo fato de que o lado esquerdo das mesmas deva sempre conter pelo menos um símbolo não-terminal:

- $\alpha \in V^*NV^*$;
- $\beta \in V^*$;
- Não se exige a validade de qualquer relação restritiva entre $|\beta|$ e $|\alpha|$.

Exemplo 2.35 A gramática $G_5 = (\{a, b, c, S, X, Y\}, \{a, b, c\}, \{S \rightarrow aXb, S \rightarrow aXa, Xa \rightarrow c, Xb \rightarrow c, X \rightarrow \epsilon\}, S)$ é irrestrita, porém não é sensível ao contexto, devido à presença das produções $Xa \rightarrow c, Xb \rightarrow c$ e $S \rightarrow \epsilon$. As gramáticas G_1, G_2, G_3 e G_4 são todas irrestritas. □

Diferentemente do que ocorre com as gramáticas sensíveis ao contexto, as formas sentenciais obtidas pela aplicação das regras de uma gramática irrestrita não possuem comprimento necessariamente crescente: elas podem aumentar ou diminuir de comprimento, conforme as produções que forem aplicadas.

Às linguagens geradas por gramáticas do tipo 0 dá-se o nome de **recursivamente enumeráveis**, **irrestritas** ou, ainda, do **tipo 0**. Como se pode perceber pela definição, toda gramática do tipo 1 também é uma gramática do tipo 0, porém o contrário não é verdadeiro. Em particular, aquelas que possuem produções em que $|\beta| < |\alpha|$.

Logo, toda linguagem sensível ao contexto pode também ser considerada uma linguagem irrestrita. Demonstra-se, por outro lado, que existem linguagens irrestritas que

não podem ser geradas por qualquer gramática sensível ao contexto. Portanto, as linguagens sensíveis ao contexto constituem subconjunto próprio das linguagens irrestritas.

Resumidamente, a Hierarquia de Chomsky original estabelece que:

1. Toda linguagem do tipo $i, 0 \leq i \leq 3$ é gerada por uma gramática do tipo i;

2. A classe das linguagens do tipo $i, 1 \leq i \leq 3$ está incluída propriamente na classe das linguagens $i - 1$. Conseqüentemente, toda linguagem do tipo $i, 1 \leq i \leq 3$ é também uma linguagem do tipo $i - 1$.

Conforme o trabalho original de Chomsky, no caso das linguagens do tipo 2 (livres de contexto), as regras acima são válidas apenas para linguagens que não contêm a cadeia vazia. Como esse texto considera L do tipo 1 se e apenas se $L - \{\epsilon\}$ for gerada por alguma gramática do tipo 1, então as regras acima são sempre verdadeiras, sem qualquer restrição.

Exceto por esse detalhe, a hierarquia de inclusão própria das linguagens definida por Chomsky pode ser representada graficamente através da Figura 2.7:

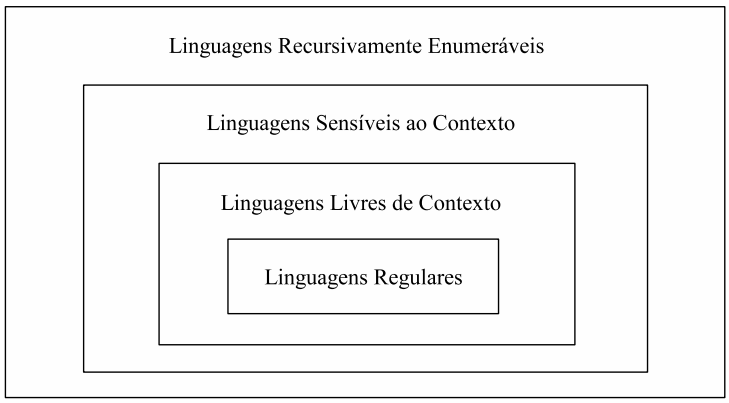

Figura 2.7: Hierarquia de Chomsky

A relação entre as gramáticas dos tipos 0, 1, 2 e 3 pode ser resumida da seguinte forma:

- Toda gramática do tipo 3 (linear) é também do tipo 2 (livre de contexto);

- Nem toda gramática do tipo 2 é também do tipo 1 (sensível ao contexto). São do tipo 1 apenas aquelas que não possuem produções $\alpha \rightarrow \beta$ em que $\beta = \epsilon$;

- Toda gramática do tipo 1 é também do tipo 0 (irrestrita).

Os critérios que levaram Chomsky a classificar as linguagens em quatro classes diferentes foram baseados nos seguintes aspectos (os quais, na verdade, estão intimamente relacionados um ao outro). Em todos os casos a seguir, constatam-se semelhanças para linguagens pertencentes a uma mesma classe e diferenças para linguagens pertencentes a classes diferentes:

1. O conjunto das propriedades exibidas pela linguagem;

2. As características estruturais mais significativas exibidas pela linguagem;

3. O modelo de reconhecedor mais simples necessário ao seu reconhecimento.

Do ponto de vista estrutural, as linguagens estritamente livres de contexto, ou seja, as livres de contexto que não podem ser geradas por gramáticas lineares à esquerda ou à direita, distinguem-se das linguagens regulares por exibirem características de **aninhamento**, como é o caso da estruturação em blocos e comandos, ou ainda do uso de parênteses em expressões aritméticas nas linguagens de programação tradicionais. Gramáticas livres de contexto possibilitam a formalização gramatical de tais aninhamentos, ao passo que gramáticas lineares não oferecem tal possibilidade.

As linguagens estritamente sensíveis ao contexto, ou seja, as sensíveis ao contexto que não podem ser geradas por gramáticas livres de contexto, distinguem-se das linguagens livres de contexto por conterem sentenças as quais, por sua vez, exibem construções, ou conjuntos de símbolos, cuja validade é vinculada à presença de construções "associadas" em regiões distintas da mesma cadeia. Tais vinculações (como, por exemplo, aquelas que existem entre a declaração e o uso de variáveis) são características de linguagens do tipo 1, e recebem a designação de **sensibilidade** ou **dependência** de contexto.

As linguagens estritamente irrestritas, ou seja, as irrestritas que não podem ser geradas por gramáticas sensíveis ao contexto, distinguem-se das linguagens sensíveis ao contexto por apresentarem características de **indecidibilidade** que lhes são particulares: demonstra-se, por exemplo, nos próximos capítulos, que não há solução para o problema genérico de se decidir se uma cadeia qualquer pertence ou não à linguagem gerada por uma gramática quando esta for do tipo 0. No entanto, o mesmo problema, quando aplicado a gramáticas do tipo 1, pode sempre ser resolvido.

Do ponto de vista analítico, verifica-se que cada classe de linguagens mencionada demanda, de acordo com sua complexidade, classes de reconhecedores progressivamente mais poderosos, porém sempre aderentes ao modelo genérico anteriormente apresentado. Por outro lado, é possível demonstrar formalmente a equipotência das várias classes de linguagens com os correspondentes modelos específicos de reconhecedores e gramáticas, caracterizando cada modelo como sendo necessário e suficiente para a análise ou síntese de linguagens pertencentes a cada classe.

Naturalmente, sempre será possível reconhecer linguagens de tipo i com reconhecedores apropriados para linguagens do tipo $i-1, 1 \leq i \leq 3$. No entanto, para efeitos práticos, o custo e a complexidade exibida pelos modelos mais complexos não justificam a sua utilização, dada a sensível redução de complexidade observada quando se adotam modelos equivalentes mais simples.

São quatro as particularizações do modelo geral de reconhecedor que serão estudadas em detalhes nos próximos capítulos: os autômatos finitos, os autômatos de pilha, as Máquinas de Turing com limitação de memória e, finalmente, as Máquinas de Turing sem limitação de memória.

A associação das quatro classes de linguagens definidas pela Hierarquia de Chomsky, com os respectivos modelos de autômato necessários ao seu reconhecimento, está resumida na Tabela 2.1.

Tipo	**Classe de linguagens**	**Modelo de gramática**	**Modelo de reconhecedor**
0	Recursivamente enumeráveis	Irrestrita	Máquina de Turing
1	Sensíveis ao contexto	Sensível ao contexto	Máquina de Turing com fita limitada
2	Livres de contexto	Livre de contexto	Autômato de pilha
3	Regulares	Linear (direita ou esquerda)	Autômato finito

Tabela 2.1: Linguagens, gramáticas e reconhecedores

As propriedades, as características estruturais e também os modelos de reconhecimento mais adequados para cada uma das classes de linguagens definidas pela Hierarquia de Chomsky serão estudados em detalhes nos capítulos seguintes.

3 Linguagens Regulares

Neste capítulo serão estudadas as linguagens regulares e as diversas formas através das quais elas podem ser geradas ou reconhecidas:

- Gramáticas regulares;
- Conjuntos regulares (mais usualmente representados através das expressões regulares);
- Autômatos finitos.

Em primeiro lugar, serão apresentados cada um desses formalismos. Depois, mostrada a equivalência entre os mesmos quanto à classe de linguagens que são capazes de representar — as linguagens regulares —, e serão apresentados também algoritmos que permitem efetuar a obtenção de uma representação a partir de cada uma das outras.

Especial ênfase será dedicada ao estudo dos autômatos finitos: serão introduzidas as notações e os conceitos, sempre relacionando esse tipo de reconhecedor ao modelo geral, apresentado no capítulo anterior. Também será apresentada uma introdução aos transdutores finitos.

Posteriormente, serão consideradas as propriedades exibidas pelas linguagens do tipo 3 em suas diversas representações. Em particular, o "Pumping Lemma" das linguagens regulares, muito útil na demonstração de que certas linguagens não são regulares, e também para demonstrar diversos outros importantes teoremas relativos às linguagens desta classe. Outra fundamental propriedade refere-se à existência de um autômato mínimo (e único) para cada linguagem regular considerada, o qual pode ser obtido conforme o método prático apresentado.

Em seguida, serão analisadas algumas valiosas operações que preservam a regularidade das linguagens do tipo 3, e que, por esse motivo, também são úteis para determinar a categoria de Chomsky a que uma linguagem possa pertencer.

Finalmente, serão apresentadas e demonstradas, no último item deste capítulo, algumas questões decidíveis referentes à classe das linguagens regulares.

3.1 Gramáticas Regulares

As gramáticas lineares à direita ou à esquerda foram apresentadas anteriormente na seção 2.6. Trata-se de gramáticas cujas regras $\alpha \rightarrow \beta$ atendem às seguintes condições:

- $\alpha \in N$;
- $\beta \in (\Sigma \cup \{\epsilon\})(N \cup \{\epsilon\})$ se linear à direita, ou
  $\beta \in (N \cup \{\epsilon\})(\Sigma \cup \{\epsilon\})$ se linear à esquerda.

Demonstra-se que as gramáticas lineares à esquerda ou à direita geram exatamente a mesma classe de linguagens. Portanto, é indiferente o emprego de uma ou outra dessas

duas variantes de gramática, já que ambas possuem a mesma capacidade de representação de linguagens.

Por esse motivo, as gramáticas lineares à direita ou à esquerda são também denominadas **gramáticas regulares**. Este termo serve para designar ambos os tipos de gramática linear. As linguagens geradas por gramáticas regulares recebem o nome de **linguagens regulares**. O Teorema 3.1 estabelece a equivalência entre gramáticas lineares, à direita e à esquerda.

Teorema 3.1 (Linear à direita ⇔ linear à esquerda) *Se G_1 é uma gramática linear à direita, então existe uma gramática linear à esquerda G_2 tal que $L(G_1) = L(G_2)$, e vice-versa.*

Justificativa Considere-se $L^R = L^R(G_1)$, o reverso da linguagem definida por G_1.[1]

Considere-se a gramática linear à direita G' tal que $L^R = L(G')$. Considere-se também G'', obtida a partir de G', de tal forma que as cadeias β de comprimento não-unitário de G' sejam invertidas em G'', conforme o Algoritmo 3.1:

Algoritmo 3.1 (Linguagem reversa) *Obtenção de uma gramática linear à esquerda que gera L^R a partir de uma gramática linear à direita que gera L.*

- Entrada: uma gramática linear à direita $G' = (V, \Sigma, P', S)$;
- Saída: uma gramática linear à esquerda $G'' = (V, \Sigma, P'', S)$, tal que $L(G'') = L(G')^R$;
- Método:

  1. $P'' \leftarrow \varnothing$;
  2. Se $\alpha \rightarrow \beta \in P', \beta \in (\Sigma \cup N \cup \{\epsilon\})$, então $\alpha \rightarrow \beta \in P''$;
  3. Se $\alpha \rightarrow \beta \in P', \beta \in (\Sigma N)$, então $\alpha \rightarrow \beta^R \in P''$;

Com G'' construída dessa forma, é possível demonstrar que $L(G'') = L^R(G') = L(G_1)$. Como G'' é, por construção, linear à esquerda, segue que $G'' = G_2$ e $L(G_1) = L(G_2)$. A demonstração no sentido oposto é obtida de forma análoga.

As gramáticas lineares à direita geram formas sentenciais em que o símbolo não-terminal é sempre o último símbolo das mesmas ($\gamma X, \gamma \in \Sigma^*, X \in N$). Portanto, as sentenças da linguagem vão sendo construídas através da inserção de novos símbolos terminais sempre à direita das formas sentenciais, imediatamente antes do símbolo não-terminal (daí o nome "linear à direita").

O algoritmo deste teorema mostra como obter uma gramática linear à esquerda que gera a mesma linguagem de uma dada gramática linear à direita. De fato, a linguagem gerada é a mesma, com a única diferença de que agora as sentenças são construídas no sentido oposto, ou seja, através da inserção de símbolos terminais sempre à esquerda das formas sentenciais, imediatamente depois do símbolo não-terminal, que agora é o primeiro das formas sentenciais geradas ($X\gamma, \gamma \in \Sigma^*, X \in N$). Só muda, portanto, o sentido em que as sentenças são geradas.

[1] Se L é regular, então, conforme demonstrado no Teorema 3.22, L^R é também regular. E, de acordo com o Teorema 3.6, existe uma gramática linear à direita G' que gera L^R.

Dada uma gramática linear à direita G_1, o procedimento completo consiste, portanto, nos passos descritos no Algoritmo 3.2:

Algoritmo 3.2 (Linear à direita ⇔ esquerda) *Obtenção de gramática linear à esquerda G_2 equivalente a uma gramática linear à direita G_1.*

- Entrada: uma gramática linear à direita G_1;
- Saída: uma gramática linear à esquerda G_2, tal que $L(G_2) = L(G_1)$;
- Método:

  1. Determinar $L(G_1)$;
  2. Determinar $L(G_1)^R$;
  3. Obter uma gramática linear à direita G' tal que $L(G') = L(G_1)^R$;
  4. Transformar G' em G_2 através da inversão das regras, conforme o Algoritmo 3.1.

■

Exemplo 3.1 Considere a gramática linear à direita G_1 definida a seguir:

$$\begin{array}{lcl} S & \rightarrow & aS \\ S & \rightarrow & bS \\ S & \rightarrow & P \\ P & \rightarrow & cQ \\ Q & \rightarrow & cR \\ R & \rightarrow & dR \\ R & \rightarrow & d \end{array}$$

$L(G_1)$ corresponde ao conjunto das cadeias w sobre $\{a, b, c, d\}$ tais que:

1. w começa com zero ou mais símbolos a ou b;
2. w continua com exatamente dois símbolos c;
3. w termina com um ou mais símbolos d.

Logo, $L^R(G_1)$ é tal que:

1. w começa com um ou mais símbolos d.
2. w continua com exatamente dois símbolos c;
3. w termina com zero ou mais símbolos a ou b;

Uma gramática linear à direita G' tal que $L(G') = L^R(G_1)$ é:

$$\begin{array}{lcl} S' & \rightarrow & dS' \\ S' & \rightarrow & dP' \\ P' & \rightarrow & cQ' \end{array}$$

$$\begin{array}{lcl} Q' & \rightarrow & cR' \\ R' & \rightarrow & aR' \\ R' & \rightarrow & bR' \\ R' & \rightarrow & \epsilon \end{array}$$

A gramática G_2, obtida a partir de G' pela aplicação do Algoritmo 3.1, é:

$$\begin{array}{lcl} S'' & \rightarrow & S''d \\ S'' & \rightarrow & P''d \\ P'' & \rightarrow & Q''c \\ Q'' & \rightarrow & R''c \\ R'' & \rightarrow & R''a \\ R'' & \rightarrow & R''b \\ R'' & \rightarrow & \epsilon \end{array}$$

Como é fácil observar, G_2 é linear à esquerda e $L(G_2) = L(G_1)$. Por exemplo, considerem-se as derivações da sentença $abaccdd$, respectivamente em G_1 e G_2:

- $S \overset{G_1}{\Rightarrow} aS \overset{G_1}{\Rightarrow} abS \overset{G_1}{\Rightarrow} abaS \overset{G_1}{\Rightarrow} abaP \overset{G_1}{\Rightarrow} abacQ \overset{G_1}{\Rightarrow} abaccR \overset{G_1}{\Rightarrow} abaccdR \overset{G_1}{\Rightarrow} abaccdd$
- $S'' \overset{G_2}{\Rightarrow} S''d \overset{G_2}{\Rightarrow} P''dd \overset{G_2}{\Rightarrow} Q''cdd \overset{G_2}{\Rightarrow} R''ccdd \overset{G_2}{\Rightarrow} R''accdd \overset{G_2}{\Rightarrow} R''baccdd \overset{G_2}{\Rightarrow} R''abaccd \overset{G_2}{\Rightarrow} abaccdd$

□

Alguns autores consideram as seguintes extensões na definição das regras $\alpha \rightarrow \beta$ de gramáticas lineares à direita e à esquerda:

- $\alpha \in N$;
- $\beta \in \Sigma^*(N \cup \{\epsilon\})$ se linear à direita, ou
  $\beta \in (N \cup \{\epsilon\})\Sigma^*$ se linear à esquerda.

Nessas extensões, admite-se uma quantidade qualquer de símbolos terminais no lado direito das produções gramaticais, e não no máximo um, como foi estabelecido na definição original. Tais extensões em nada alteram a classe de linguagens representáveis por esses tipos de gramáticas, constituindo o seu uso mera conveniência. O Teorema 3.2 traz a demonstração dessa equivalência.

Teorema 3.2 (Desmembramento de uma gramática linear à direita) *Se G_1 é uma gramática composta apenas de produções do tipo $\alpha \rightarrow \beta, \alpha \in N, \beta \in \Sigma^*(N \cup \{\epsilon\})$, então existe uma gramática equivalente G_2 composta apenas de produções do tipo $\alpha \rightarrow \beta, \alpha \in N, \beta \in (\Sigma \cup \{\epsilon\})(N \cup \{\epsilon\})$.*

Justificativa G_2 pode ser obtida a partir de G_1 pelo Algoritmo 3.3, o qual substitui as regras $\alpha \rightarrow \beta, \beta \in \Sigma^* N$, por um conjunto equivalente de novas regras $\alpha' \rightarrow \beta', \beta' \in (\Sigma \cup \{\epsilon\})(N \cup \{\epsilon\})$.[2]

Algoritmo 3.3 ($\boldsymbol{\alpha \rightarrow \beta, \beta \in \Sigma^*(N \cup \{\epsilon\}) \Rightarrow \alpha \rightarrow \beta, \beta \in (\Sigma \cup \{\epsilon\})(N \cup \{\epsilon\})}$) *Desmembramento das produções de uma gramática linear à direita, na forma $\alpha \rightarrow \beta$,*

[2]Tal tipo de gramática recebe, em alguns textos, a denominação de **gramática linear unitária à direita**.

$\beta \in \Sigma^*(N \cup \{\epsilon\})$, *em conjuntos de produções equivalentes, na forma* $\alpha \to \beta$, $\beta \in (\Sigma \cup \{\epsilon\})(N \cup \{\epsilon\})$.

- Entrada: uma gramática linear à direita $G_1 = (V_1, \Sigma, P_1, S_1)$, cujas produções são da forma $\alpha \to \beta, \beta \in \Sigma^*(N \cup \{\epsilon\})$;
- Saída: uma gramática linear à direita $G_2 = (V_2, \Sigma, P_2, S_2)$, tal que $L(G_2) = L(G_1)$ e cujas produções são todas da forma $\alpha \to \beta, \beta \in (\Sigma \cup \{\epsilon\})(N \cup \{\epsilon\})$;
- Método:

  1. $N_2 \leftarrow N_1$;
  2. $P_2 \leftarrow \varnothing$;
  3. Para cada $\alpha \to \beta \in P_1, \beta = \sigma_1 ... \sigma_n K$, com $K \in N \cup \{\epsilon\}$ e $n \geq 0$, faça:
     - Se $\beta = \epsilon$, $\beta \in \Sigma$, $\beta \in N$, ou $\beta \in \Sigma N$, então $P_2 \leftarrow P_2 \cup \{\alpha \to \beta\}$
     - Se $|\beta| \geq 2$ e $\beta \in \Sigma^*$, ou seja, se $\beta \in \Sigma\Sigma\Sigma^*$, então:
       * $N_2 \leftarrow N_2 \cup \{Y_1, Y_2, ..., Y_{n-1}\}$
       * $P_2 \leftarrow P_2 \cup \{\alpha \to \sigma_1 Y_1,\ Y_1 \to \sigma_2 Y_2\ , ..., Y_{n-2} \to \sigma_{n-1}\ Y_{n-1},\ Y_{n-1} \to \sigma_n\}$
     - Se $|\beta| \geq 3$ e $\beta \in \Sigma^* N$, ou seja, se $\beta \in \Sigma\Sigma\Sigma^* N$, então:
       * $N_2 \leftarrow N_2 \cup \{X_1, X_2, ..., X_{n-1}\}$
       * $P_2 \leftarrow P_2 \cup \{\alpha \to \sigma_1 X_1,\ X_1 \to \sigma_2 X_2, ..., X_{n-2} \to \sigma_{n-1} X_{n-1},\ X_{n-1} \to \sigma_n K\}$

Algoritmo semelhante pode ser facilmente desenvolvido para o caso das gramáticas lineares à esquerda, de forma a obter uma gramática equivalente cujas regras sejam do tipo $\alpha \to \beta, \alpha \in N, \beta \in (N \cup \{\epsilon\})(\Sigma \cup \{\epsilon\})$.[3] ■

Exemplo 3.2 Considere-se a gramática G_1:

$$\begin{array}{rcl} S_1 & \to & abcdP \\ P & \to & efP \\ P & \to & Q \\ Q & \to & g \end{array}$$

A aplicação do Algoritmo 3.3 resulta na gramática G_2:

$$S_2 \quad \to \quad aP_1$$

[3]Também conhecida como **gramática linear unitária à esquerda**.

$$\begin{array}{rcl} P_1 & \rightarrow & bP_2 \\ P_2 & \rightarrow & cP_3 \\ P_3 & \rightarrow & dP \\ P & \rightarrow & eP_4 \\ P_4 & \rightarrow & fP \\ P & \rightarrow & Q \\ Q & \rightarrow & g \end{array}$$

A título de ilustração, considerem-se as derivações da sentença $abcdefg$, respectivamente em G_1 e G_2:

- $S_1 \stackrel{G_1}{\Rightarrow} abcdP \stackrel{G_1}{\Rightarrow} abcdefP \stackrel{G_1}{\Rightarrow} abcdefQ \stackrel{G_1}{\Rightarrow} abcdefg$
- $S_2 \stackrel{G_2}{\Rightarrow} aP_1 \stackrel{G_2}{\Rightarrow} abP_2 \stackrel{G_2}{\Rightarrow} abcP_3 \stackrel{G_2}{\Rightarrow} abcdP \stackrel{G_2}{\Rightarrow} abcdeP_4 \stackrel{G_2}{\Rightarrow} abcdefP \stackrel{G_2}{\Rightarrow} abcdefQ \stackrel{G_2}{\Rightarrow} abcdefg$

□

3.2 Conjuntos e Expressões Regulares

Conjuntos e expressões regulares são notações alternativas utilizadas para representar a classe de linguagens mais simples que se conhece: a classe das linguagens regulares, a mais restrita dentro da Hierarquia de Chomsky.

Conjuntos regulares sobre um alfabeto finito Σ são linguagens definidas recursivamente da seguinte forma:

1. $\varnothing$ é um conjunto regular sobre Σ;
2. $\{\epsilon\}$ é um conjunto regular sobre Σ;
3. $\{\sigma\}, \forall \sigma \in \Sigma$, é um conjunto regular sobre Σ;

   Se X e Y são conjuntos regulares sobre Σ, então também são conjuntos regulares sobre Σ:

4. (X);
5. $X \cup Y$;
6. $X \cdot Y$, também denotado XY;
7. X^*.

Diz-se que um determinado subconjunto de Σ^* é um conjunto regular se ele puder ser formulado através do uso combinado dessas regras apenas.

Exemplo 3.3 Seja $L = \{0^m 1^n \mid m \geq 0, n \geq 0\}$ sobre $\Sigma = \{0, 1\}$. A linguagem L é formada por sentenças em que a concatenação de um número arbitrário de símbolos "0" (incluindo nenhum) se concatena com a concatenação de um número também arbitrário de símbolos "1" (incluindo nenhum):

$$L = \{\epsilon, 0, 1, 00, 01, 11, ...\}$$

Considerem-se as linguagens sobre Σ, abaixo discriminadas:

$$\begin{array}{rcl} L_1 & = & \{0\} \\ L_2 & = & \{1\} \end{array}$$

$$\begin{aligned} L_3 &= \{0^i \mid i \geq 0\} \\ L_4 &= \{1^i \mid i \geq 0\} \\ L_5 &= \{0^p 1^q \mid p \geq 0, q \geq 0\} \end{aligned}$$

Os conjuntos L_1 e L_2 são conjuntos regulares sobre Σ, por definição. L_3 e L_4 são obtidos a partir de L_1 e L_2, respectivamente, pela aplicação da operação fechamento reflexivo e transitivo, ou seja, $L_3 = L_1^*$ e $L_4 = L_2^*$. Por sua vez, o conjunto $L_5 = L$ pode ser expresso pela concatenação dos conjuntos L_3 e L_4, isto é, $L_5 = L_3 L_4$. Dessa maneira, demonstra-se que $L = \{0^m 1^n \mid m \geq 0, n \geq 0\}$ é um conjunto regular sobre $\{0, 1\}$. Na notação dos conjuntos regulares, L pode ser denotado por $\{0\}^*\{1\}^*$. □

Exemplo 3.4 A linguagem N formada pelos números naturais decimais é um conjunto regular sobre o alfabeto dos algarismos arábicos e pode ser representada através do seguinte conjunto regular:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$$

Se $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, então $N = DD^*$.

O conjunto R dos números reais decimais sem sinal é um conjunto regular sobre $D \cup \{.\}$, e pode ser representado por:

$$DD^*.D^* \cup D^*.DD^*$$

Observe-se que a definição acima inclui números iniciando ou terminando com o caractere "." (como, por exemplo, .315 ou 47.), porém exclui da linguagem a cadeia ".". O conjunto P dos números em ponto flutuante com expoente (denotado por "E") e sinal opcional ("+" ou "−") pode ser representado por:

$$\{+, -, \epsilon\}(DD^*.D^* \cup D^*.DD^*)\{E\}\{+, -, \epsilon\}DD^*$$

Assim, por exemplo, $27 \in N, 915.4 \in R$ e $-211.56E + 3 \in P$. Deve-se notar que $N \subset R \subset P, P \neq R$ e $R \neq N$. □

A definição de conjuntos regulares envolve a aplicação de três operações já estudadas para os conjuntos: união, concatenação e fechamento reflexivo e transitivo. No caso do fechamento, no entanto, cabem algumas observações adicionais válidas para o caso em que o seu operando seja não apenas um alfabeto, conforme anteriormente mencionado, mas eventualmente uma linguagem, como ocorre na definição acima.

Seja L uma linguagem qualquer, e considerem-se as novas linguagens L^* e L^+ obtidas pela aplicação, respectivamente, das operações de fechamento reflexivo e transitivo e do fechamento transitivo sobre L. Neste caso, deve-se observar que, diferentemente do que ocorre com os alfabetos, as seguintes identidades são verdadeiras:

- $L^+ = L^*$ se $\epsilon \in L$
- $L^+ = L^* - \{\epsilon\}$ se $\epsilon \notin L$

Como alternativa para a representação dos conjuntos regulares, visando obter maior concisão e facilidade de manipulação, Kleene desenvolveu, na década 1950, a notação das **expressões regulares** ([60]). Da mesma forma como ocorre para os conjuntos regulares, as expressões regulares sobre um alfabeto Σ podem também ser definidas recursivamente como segue:

1. $\varnothing$ é uma expressão regular e denota o conjunto regular $\varnothing$;
2. ϵ é uma expressão regular e denota o conjunto regular $\{\epsilon\}$;
3. Cada $\sigma, \sigma \in \Sigma$, é uma expressão regular e denota o conjunto regular $\{\sigma\}, \sigma \in \Sigma$;

Se x e y são expressões regulares sobre Σ que denotam, respectivamente, os conjuntos regulares X e Y, então:

4. (x);
5. $x \mid y$ ou $x + y$;
6. $x \cdot y$ ou xy;
7. x^*

também são expressões regulares e denotam, respectivamente, os conjuntos regulares $X, X \cup Y, XY$ e X^*. Note-se a eliminação, nas expressões regulares, do uso dos símbolos "{" e "}", bem como a substituição do operador "∪" pelo operador "+" ou "|" (a critério de cada autor). Visando tornar ainda mais cômoda a utilização das expressões regulares, admite-se a eliminação dos pares de parênteses envolvendo sub-expressões que contenham seqüências exclusivas de operadores, de união ou de concatenação, uma vez que se trata de operações associativas. Além disso, são designadas precedências distintas para as três operações, reduzindo ainda mais a necessidade de emprego de parênteses nas expressões regulares. A Tabela 3.1 resume esses aspectos.

Precedência	**Operador**	**Representação**
Mais alta	Fechamento	x^*
Intermediária	Concatenação	$x \cdot y$ ou xy
Mais baixa	União	$x \mid y$ ou $x + y$

Tabela 3.1: Precedência dos operadores nas expressões regulares

Os parênteses são empregados para modificar localmente a precedência ou a associatividade pré-definida dos operadores, assim como ocorre nas expressões aritméticas tradicionais da matemática.

Exemplo 3.5 A expressão regular $(ab \mid c^*) = ((ab) \mid c^*) = ((ab) \mid (c^*))$ representa o conjunto $\{ab, \epsilon, c, cc, ccc...\}$. A expressão regular $a(b \mid c)^*$ representa o conjunto $\{a, ab, ac, abc, abb, acc, ...\}$. Finalmente, $(ab \mid c)^*$ representa o conjunto $\{\epsilon, ab, c, abc, cab, abab, cc, ...\}$. □

Uma abreviação muito comum consiste na substituição da expressão regular xx^* por x^+, denotando com isso o conjunto regular correspondente ao fechamento transitivo de X (que é composto por todas as cadeias de comprimento maior ou igual a 1 que podem ser construídas sobre o conjunto X).

Exemplo 3.6 Considerem-se o alfabeto $\Sigma = \{a, b, c, d\}$ e os dois subconjuntos $A = \{a\}, B = \{b, c\}$. A seguir são apresentadas diferentes linguagens sobre Σ, definidas através da notação dos conjuntos e das expresões regulares:

- Sentenças que possuem no mínimo um símbolo a:
  $\Sigma^* A \Sigma^*$ ou $(a \mid b \mid c \mid d)^* a (a \mid b \mid c \mid d)^*$
- Sentenças que possuem exatamente dois símbolos a:
  $(\Sigma - A)^* A (\Sigma - A)^* A (\Sigma - A)^*$ ou $(b \mid c \mid d)^* a (b \mid c \mid d)^* a (b \mid c \mid d)^*$

- Sentenças que possuem um número par de símbolos a:
  $((\Sigma - A)^* A(\Sigma - A)^* A(\Sigma - A)^*)^*$ ou $((b \mid c \mid d)^* a(b \mid c \mid d)^* a(b \mid c \mid d)^*)^*$
- Sentenças que são iniciadas com o símbolo a e terminam com o símbolo b ou c:
  $A\Sigma^* B$ ou $a(a \mid b \mid c \mid d)^*(b \mid c)$
- Sentenças contendo apenas os símbolos a, b, c, com no mínimo um símbolo:
  $(A \cup B)^+$ ou $(a \mid b \mid c)^+$
- Sentenças formadas por símbolos do alfabeto $\{a, b, c, d\}$ contendo uma (e somente uma) subcadeia constituída por um símbolo do conjunto A e dois (e somente dois) do conjunto B, nesta ordem:
  $((\Sigma - A) - B)^* ABB((\Sigma - A) - B)^*$ ou $d^* a(b \mid c)(b \mid c)d^*$

□

Exemplo 3.7 Utilizando-se a notação das expressões regulares, a linguagem $L = \{0^m 1^n \mid m \geq 0, n \geq 0\}$ pode ser reescrita como $((0)^*(1)^*)$, ou, simplesmente, 0^*1^*. Para $m \geq 0$ e $n \geq 1$, a expressão correspondente seria 0^*11^*. Note-se que $0^*11^* = 0^*1^*1 = 0^*1^+$. □

A seguir serão apresentadas algumas relações de identidade válidas para as expressões regulares, as quais podem ser demonstradas sem dificuldade. Sejam x, y, z três expressões regulares quaisquer. Então:

- $x \mid y = y \mid x$
- $\varnothing^* = \epsilon$
- $x \mid (y \mid z) = (x \mid y) \mid z$
- $x(yz) = (xy)z$
- $x(y \mid z) = xy \mid xz$
- $(x \mid y)z = xz \mid yz$
- $x\epsilon = \epsilon x = x$
- $x\varnothing = \varnothing x = \varnothing$
- $\epsilon\varnothing = \varnothing\epsilon = \varnothing$
- $x^* = x \mid x^*$
- $(x^*)^* = x^*$
- $x \mid x = x$
- $x \mid \varnothing = x$
- $(xy)^* x = x(yx)^*$

Por se tratar de uma notação concisa, que dispensa o uso da notação dos conjuntos e o emprego de símbolos não-terminais para a definição de linguagens, mas que, ao mesmo

tempo, permite a plena representação dos conjuntos regulares, as expressões regulares são bastante utilizadas em áreas que abrangem desde a especificação de linguagens de programação e de comandos, entre outras, até a entrada de dados em editores de texto, programas de busca, análise de padrões etc.

As linguagens regulares foram definidas no capítulo anterior como a classe das linguagens geradas por gramáticas lineares, à esquerda ou à direita. No presente item foram apresentados os conjuntos regulares e a notação mais comumente utilizada para a sua representação, as expressões regulares.

3.3 Autômatos Finitos

Da mesma forma como ocorre com as expressões regulares e com as gramáticas lineares à direita, os **autômatos finitos** também possibilitam a formalização das linguagens regulares, ou seja, das linguagens do tipo 3. No entanto, diferentemente daquelas notações, que constituem dispositivos de geração de sentenças, os autômatos finitos são dispositivos de aceitação de sentenças e constituem um caso particular do modelo geral de reconhecedores apresentado no Capítulo 2.

A seguir serão introduzidas as notações, as convenções, as características de operação e algumas variantes mais comuns dos autômatos finitos. O item seguinte mostra que a classe de linguagens por eles aceita coincide exatamente com a classe das linguagens definidas pelos conjuntos regulares e também pelas gramáticas lineares à direita.

Os autômatos finitos podem ser determinísticos ou não-determinísticos, apresentando ou não transições em vazio. Conforme analisado mais adiante, a eventual presença de não-determinismos e/ou de transições em vazio não altera a classe de linguagens aceita pelos autômatos finitos. Por motivos estritamente didáticos, serão introduzidos inicialmente os autômatos finitos determinísticos, sendo feitas extensões posteriores para contemplar a existência de não-determinismos e de transições em vazio.

Os autômatos finitos correspondem à instância mais simples do modelo geral de reconhecedores apresentado na Seção 2.5. As suas principais particularidades em relação ao modelo geral são:

1. Inexistência de memória auxiliar;

2. Utilização do cursor da fita de entrada apenas para leitura de símbolos, não havendo operações de escrita sobre a fita;

3. Movimentação do cursor de leitura em apenas um sentido, da esquerda para a direita;

4. A fita de entrada possui comprimento limitado, suficiente apenas para acomodar a cadeia a ser analisada.

Os autômatos finitos podem ser representados em notação algébrica ou através de diagramas de transição de estados, introduzidos a seguir, mais adequados à sua visualização.

Autômatos Finitos Determinísticos

Algebricamente, um autômato finito determinístico M pode ser definido como uma quíntupla:

$$M = (Q, \Sigma, \delta, q_0, F)$$

- Q é um conjunto finito de estados;
- Σ é um alfabeto (finito e não-vazio) de entrada;
- δ é uma função de transição, $\delta : Q \times \Sigma \rightarrow Q$;
- q_0 é o estado inicial, $q_0 \in Q$;
- F é um conjunto de estados finais, $F \subseteq Q$.

A máquina de estados de um autômato finito, também denominada **controle finito**, é definida pelo conjunto de estados Q e pela função de transição δ, que associa pares ordenados do tipo *(estado corrente, entrada corrente)* com um novo estado a ser assumido pelo autômato quando da aplicação da transição.

Deve-se notar que a função de transição δ pode ser, no caso dos autômatos finitos determinísticos, uma função total, ou seja, uma função que é definida para todos os elementos de $Q \times \Sigma$, ou ainda uma função parcial. Se total, isso implica a especificação de transições com cada um dos possíveis símbolos de entrada $\sigma \in \Sigma$ para cada um dos possíveis estados $q \in Q$ do autômato finito. Assim, se $\left|\Sigma\right| = m$ e $\left|Q\right| = n$, então o autômato finito determinístico possuirá, exatamente, $m * n$ transições distintas.

As transições de um autômato finito podem ser denotadas através de expressões do tipo $(p, \sigma) \rightarrow q$, com $p, q \in Q, \sigma \in \Sigma$. Alternativamente, pode-se explicitar a função δ, representando uma transição na forma $\delta(p, \sigma) = q$.

A utilização do termo "determinístico" para designar esse tipo de autômato finito decorre do fato de que, enquanto houver símbolos na fita de entrada, será sempre possível determinar o estado seguinte a ser assumido pelo autômato, o qual será único em todas as situações.

Em certos casos, especialmente na demonstração de alguns teoremas, torna-se conveniente estender o domínio da função δ para Σ^*, em vez de apenas Σ, conforme indicado abaixo:

- $\delta(q, \epsilon) = q$;
- $\delta(q, \sigma x) = \delta(\delta(q, \sigma), x), x \in \Sigma^*, \sigma \in \Sigma$.

Ao longo deste texto, a definição considerada para a função δ deverá variar conforme o contexto em que estiver sendo empregada.

A **configuração** de um autômato finito é definida pelo seu estado corrente e pela parte da cadeia de entrada ainda não analisada (incluindo o símbolo apontado pelo cursor). A **configuração inicial** de um autômato finito é aquela em que o estado corrente é q_0 (estado inicial) e o cursor de leitura se encontra posicionado sobre o símbolo mais à esquerda da cadeia de entrada. Uma **configuração final** é aquela em que o cursor aponta para a posição imediatamente além do último símbolo da cadeia (indicando com isso já ter ocorrido a leitura do último símbolo da cadeia de entrada), e o estado corrente pertence ao conjunto F de estados finais, especificado para o autômato. Note que ambas

as condições devem ser simultaneamente verificadas para permitir a caracterização de uma configuração como sendo, respectivamente, inicial ou final.

O autômato finito opera efetuando uma série de movimentos que o conduzem através dos seus estados a partir da configuração inicial. Pela inspeção do estado corrente e também do símbolo apontado pelo cursor, determina-se o próximo estado a ser atingido pelo autômato e avança-se o cursor de leitura uma célula para a direita. Esse processo é repetido seguidas vezes até que na fita de entrada não haja mais símbolos a serem lidos.

Quando ocorre o esgotamento da cadeia de entrada, deve-se analisar o tipo do estado corrente do autômato. Se for um estado final, diz-se que o autômato **reconheceu**, ou **aceitou**, a cadeia de entrada; se for um estado não-final, diz-se que a cadeia de entrada foi **rejeitada** pelo autômato — logo, a cadeia analisada não pertence à linguagem por ele definida.

Quando definidos através de funções de transição totais, os correspondentes autômatos finitos determinísticos sempre percorrem integralmente toda e qualquer cadeia de entrada (sobre Σ) que lhes forem apresentadas para análise. Nesses casos, portanto, a configuração final definida para um autômato finito é sempre satisfeita no que se refere ao esgotamento da cadeia de entrada, restando apenas a análise do tipo do estado atingido em tal configuração (final ou não-final) para se determinar, respectivamente, a aceitação ou a rejeição da cadeia de entrada.

Tais conceitos podem ser formalizados denotando-se a configuração de um autômato finito como um par (q, y), em que q representa o estado corrente e y a parte da cadeia de entrada ainda não analisada. A configuração inicial, com vistas ao reconhecimento de uma cadeia x, é representada como (q_0, x), e a configuração final como (q_i, ϵ), $q_i \in F$. A movimentação de um autômato de uma configuração para a configuração seguinte é denotada através do símbolo "$\vdash$":

$$(q_i, \sigma\beta) \vdash (q_j, \beta), \text{ com } q_i, q_j \in Q, \sigma \in \Sigma, \beta \in \Sigma^*, \delta(q_i, \sigma) = q_j$$

A linguagem L definida por um autômato finito M é o conjunto de todas as cadeias w sobre o alfabeto Σ que levam M da sua configuração inicial para alguma configuração final através da aplicação sucessiva de transições definidas pela função δ. Denota-se como:

$$L(M) = \{w \in \Sigma^* \mid (q_0, w) \vdash^* (q_F, \epsilon), q_F \in F\}$$

Alternativamente, conforme a extensão introduzida para a função δ, pode-se denotar o mesmo processo assim:

$$L(M) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$$

Diagramas de transição de estados são grafos orientados não-ordenados, rotulados nos vértices com os nomes dos estados e nos arcos com os símbolos do alfabeto de entrada do autômato finito. Trata-se de uma representação gráfica equivalente à notação algébrica, oferecendo porém uma melhor visualização do autômato. Nessa representação, círculos representam os estados, e arcos as transições. O estado inicial é identificado por um arco cuja extremidade inicial não é ligada a nenhum outro estado. Os estados finais são representados por círculos duplos concêntricos.

Exemplo 3.8 Seja M um autômato finito determinístico, com função de transição total, definido abaixo. Sua representação algébrica é $M = (Q, \Sigma, \delta, q_0, F)$, onde:

$$Q = \{q_0, q_1, q_2, q_3\}$$

$$
\begin{aligned}
\Sigma &= \{0,1,2\} \\
\delta &= \{(q_0,0) \to q_0, (q_0,1) \to q_1, (q_0,2) \to q_3, \\
&\quad (q_1,0) \to q_3, (q_1,1) \to q_1, (q_1,2) \to q_2, \\
&\quad (q_2,0) \to q_3, (q_2,1) \to q_3, (q_2,2) \to q_2, \\
&\quad (q_3,0) \to q_3, (q_3,1) \to q_3, (q_3,2) \to q_3\} \\
F &= \{q_1, q_2\}
\end{aligned}
$$

A Figura 3.1 mostra o diagrama de transição de estados para esse autômato:

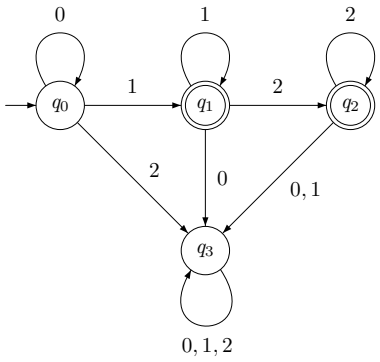

Figura 3.1: Autômato finito determinístico com função de transição total

A linguagem aceita por esse autômato finito é formada pelo conjunto de sentenças x que o levam da configuração inicial (q_0, x) até a configuração final (q_1, ϵ) ou (q_2, ϵ). A inspeção cuidadosa desse autômato finito revela que as sentenças por ele aceitas contêm, nesta ordem, uma seqüência de símbolos "0" (incluindo nenhum), seguida de uma seqüência de símbolos "1" (no mínimo um) e, finalmente, de uma seqüência de símbolos "2" (incluindo nenhum). Na notação das expressões regulares, $L(M) = 0^*1^+2^*$.

As seguintes identidades, por exemplo, são verdadeiras:

- $\delta(q_0, 00001) = q_1$
- $\delta(q_0, 0122) = q_2$
- $\delta(q_1, 12) = q_2$
- $\delta(q_2, 222) = q_2$

Esquematicamente, a configuração inicial para o reconhecimento de uma cadeia de entrada, por exemplo, a cadeia 0011222, pode ser representada conforme a figura abaixo:

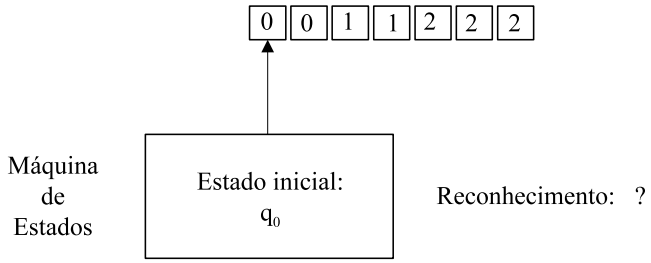

Figura 3.2: Configuração inicial

A sucessão de movimentos efetuados pelo autômato finito com essa cadeia é apresentada a seguir:

- $(q_0, 0011222) \vdash (q_0, 011222) \vdash (q_0, 11222) \vdash (q_1, 1222) \vdash (q_1, 222) \vdash (q_2, 22) \vdash (q_2, 2) \vdash (q_2, \epsilon)$

Portanto, $(q_0, 0011222) \vdash^* (q_0, \epsilon)$, e $0011222 \in L(M)$. Esquematicamente, a configuração final do autômato após o reconhecimento da cadeia de entrada 0011222 pode ser representada conforme a figura a seguir:

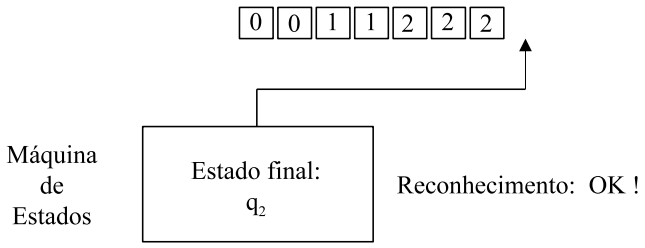

Figura 3.3: Configuração final

Analisa-se a seguir o comportamento desse autômato em relação à cadeia de entrada 0022:

- $(q_0, 0022) \vdash (q_0, 022) \vdash (q_0, 22) \vdash (q_3, 2) \vdash (q_3, \epsilon)$

Após a seqüência de movimentos acima, chega-se à configuração (q_3, ϵ), em que ocorre o esgotamento da cadeia de entrada. Como não se trata de uma configuração final, pois $q_3 \notin F$, conclui-se que a cadeia 0022 não pertence à linguagem aceita por M. □

Exemplo 3.9 O autômato finito determinístico da Figura 3.4 possui função de transição parcial, uma vez que ela não é definida para os seguintes elementos de $Q \times \Sigma$: $(q_0, 2), (q_1, 0), (q_2, 1)$ e $(q_2, 0)$.

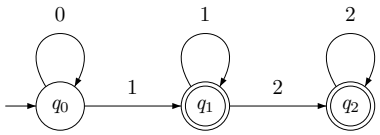

Figura 3.4: Autômato finito determinístico com função de transição parcial

Note-se que os autômatos das Figuras 3.1 e 3.4 definem a mesma linguagem. Os movimentos executados pelo autômato da Figura 3.4 com as cadeias 0011222 e 0022 são apresentados abaixo:

- $(q_0, 0011222) \vdash (q_0, 011222) \vdash (q_0, 11222) \vdash (q_1, 1222) \vdash (q_1, 222) \vdash (q_2, 22) \vdash (q_2, 2) \vdash (q_2, \epsilon)$
- $(q_0, 0022) \vdash (q_0, 022) \vdash (q_0, 22)$

No primeiro caso, a cadeia 0011222 é completamente esgotada e o autômato pára em um estado final. Logo, 0011222 é aceita pelo autômato. No segundo caso, a cadeia de entrada é apenas parcialmente consumida, e o autômato pára no estado q_o, não-final. Logo, a cadeia 0022 é rejeitada. □

Autômatos Finitos Não-Determinísticos

Apesar do elevado interesse prático que recai sobre os autômatos finitos determinísticos, uma vez que eles servem como base para a construção de programas extremamente eficientes, do ponto de vista teórico há também um interesse muito grande pelos autômatos finitos não-determinísticos, devido à maior facilidade com que eles permitem a demonstração de certos teoremas. Do ponto de vista prático, os autômatos finitos não-determinísticos são, muitas vezes, mais intuitivos do que os correspondentes autômatos finitos determinísticos, o que os torna, geralmente, mais adequados para a análise e especificação de linguagens regulares.

Um **autômato finito não-determinístico**, sem transições em vazio, difere dos autômatos finitos determinísticos pelo fato de o co-domínio da função de transição δ ser 2^Q e não simplesmente Q:

$$\delta : Q \times \Sigma \rightarrow 2^Q$$

Como conseqüência, os autômatos finitos não-determinísticos generalizam o modelo dos autômatos finitos determinísticos através das seguintes extensões:

- Introduz-se o impasse em configurações não-finais;

  Como $\varnothing \in 2^Q$, é possível não especificar transições para certas combinações de estado corrente e próximo símbolo de entrada.

- Introduz-se o não-determinismo, no sentido literal da palavra.

  Nos casos em que $\left|(\delta(q, \sigma))\right| \geq 2$, haverá mais de uma possibilidade de movimentação para o autômato finito não-determinístico na configuração corrente.

Devido principalmente à segunda condição é que autômatos deste tipo recebem a designação de não-determinísticos: havendo pelo menos uma configuração em que, para uma mesma combinação de estado corrente e símbolo de entrada, exista mais de uma alternativa de escolha do próximo estado, torna-se necessário efetuar uma opção entre elas, para permitir o prosseguimento do reconhecimento.

Note-se que esse tipo de situação nunca ocorre com os autômatos finitos determinísticos. Note-se ainda que, de acordo com a nova definição para a função δ, torna-se possível também caracterizar autômatos finitos determinísticos como casos particulares dos não-determinísticos, em que cada elemento da imagem da função δ possui exatamente um único estado.

Diz-se que um autômato finito não-determinístico **aceita** uma cadeia de entrada quando houver alguma seqüência de movimentos que o leve da configuração inicial para uma configuração final. Diferentemente do autômato finito determinístico, em que essa seqüência, se existir, é única para cada cadeia de entrada, no caso do autômato finito não-determinístico é possível que exista mais de uma seqüência que satisfaça a essa condição para uma dada cadeia de entrada. Sempre que o autômato finito não-determinístico se deparar com mais de uma possibilidade de movimentação, é feita a escolha (arbitrária) de uma das alternativas; em caso de insucesso no reconhecimento, deve-se considerar sucessivamente cada uma das demais alternativas ainda não consideradas, até o seu esgotamento; persistindo o insucesso, e esgotadas as alternativas, diz-se que o autômato **rejeita** a cadeia. A Tabela 3.2 resume esses critérios.

	Dada uma cadeia de entrada, ele:	*Aceita a cadeia de entrada se:*	*Rejeita a cadeia de entrada se:*
Autômato finito determinístico	Executa uma única seqüência de movimentos.	Pára em uma configuração final.	Pára em uma configuração não-final.
Autômato finito não-determinístico	Pode executar várias seqüências distintas de movimentos.	Pára em uma configuração final.	Pára sem conseguir atingir nenhuma configuração final.

Tabela 3.2: Aceitação e rejeição de cadeias em autômatos finitos

É importante notar que, diferentemente do que ocorre com os autômatos finitos determinísticos com função de transição total, não necessariamente os autômatos finitos não-determinísticos esgotam completamente a cadeia de entrada que lhes é oferecida para análise, mesmo que a sua função de transição seja total.

A impossibilidade de movimentação do autômato em determinadas configurações caracteriza um impasse e pode ocorrer não apenas em função do esgotamento da cadeia de entrada, como nos autômatos finitos determinísticos, mas também pela possibilidade de inexistência de transições que permitam a evolução a partir da configuração corrente devido às características da função δ. Assim, a condição que determina a aceitação de uma cadeia em um autômato finito não-determinístico deve sempre ser analisada em sua totalidade: estado final associado ao esgotamento da cadeia de entrada (lembrar que esta última condição sempre é verificada no caso dos autômatos finitos determinísticos cuja função de transição seja total).

Exemplo 3.10 Seja $M = (Q, \Sigma, \delta, \{q_0\}, F)$ um autômato finito não-determinístico:

$$\begin{aligned} Q &= \{q_0, q_1, q_2\} \\ \Sigma &= \{a, b, c\} \\ \delta &= \{(q_0, a) \rightarrow \{q_1, q_2\}, (q_1, b) \rightarrow \{q_1, q_2\}, (q_2, c) \rightarrow \{q_2\}\} \\ F &= \{q_1, q_2\} \end{aligned}$$

O diagrama de transição de estados para esse autômato, que reconhece a linguagem $ab^* \mid ab^*bc^* \mid ac^*$, ou simplesmente ab^*c^*, é apresentado na Figura 3.5.

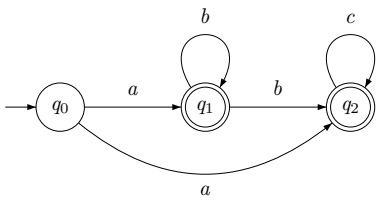

Figura 3.5: Autômato não-determinístico

Considere-se a cadeia $abbccc$ e faça-se uma simulação da operação do autômato a partir de sua configuração inicial:

$$(q_0, abbccc) \vdash (q_2, bbccc)$$

Nesta seqüência, a escolha do ramo inferior, em resposta ao símbolo de entrada a, conduz o autômato a um impasse, pois não há possibilidade de movimentação em q_2 com o símbolo b. Deve-se, então, tentar a segunda alternativa de movimentação em q_0:

$$(q_0, abbccc) \vdash (q_1, bbccc) \vdash (q_2, bccc)$$

Apesar do avanço no reconhecimento, novo impasse é atingido no estado q_2 como conseqüência da escolha efetuada em q_1 para a transição usando o símbolo b. Como não restam outras alternativas em q_0, deve-se considerar a segunda alternativa de movimentação em q_1:

$$(q_0, abbccc) \vdash (q_1, bbccc) \vdash (q_1, bccc)$$

Admitindo-se que a opção inicial de movimentação em q_1 em resposta ao símbolo b seja novamente q_2, a seguinte seqüência de movimentos conduz finalmente o autômato à sua configuração final (que, neste caso, é única):

$$(q_1, bccc) \vdash (q_2, ccc) \vdash (q_2, cc) \vdash (q_2, c) \vdash (q_2, \epsilon)$$

Seja agora a cadeia aab. Como se pode perceber, não há nenhuma seqüência de movimentos, mesmo considerando-se as duas alternativas para a transição usando a em q_0, que conduza o autômato à sua configuração final. Em um caso, o impasse é atingido no estado q_1 em decorrência do símbolo a; no outro ele ocorre em q_2, também provocado por a. Como não há mais opções decorrentes de não-determinismos a serem consideradas, conclui-se que essa cadeia não pertence à linguagem definida pelo autômato. □

Os autômatos não-determinísticos operam pela busca exaustiva de seqüências que possibilitem o reconhecimento das cadeias que lhes são apresentadas. Do ponto de vista teórico, admite-se que, toda vez em que houver mais de uma possibilidade de movimentação, o autômato finito não-determinístico se "desdobra" na quantidade correspondente de instâncias paralelas, cada qual prosseguindo à análise de forma autônoma, a partir da configuração corrente, e através de opções distintas de movimentação. Desse modo, a aceitação (ou rejeição) da sentença por um autômato finito não-determinístico deve ser analisada em função das últimas configurações atingidas por todas as suas instâncias; se pelo menos uma delas tiver atingido uma configuração final, a cadeia terá sido aceita; caso contrário, será rejeitada.

Do ponto de vista prático, no entanto, não é econômico realizar o modelo teórico de operação paralela, acima apresentado, em máquinas seqüenciais. Em vez disso, autômatos finitos não-determinísticos costumam ser implementados, dentro do modelo seqüencial de processamento, através de mecanismos de busca exaustiva e de *"backtracking"*, percorrendo-se todos os possíveis caminhos que os conduzam (ou não) a configurações finais. Por esse motivo, os autômatos finitos não-determinísticos não costumam ter muito interesse prático, uma vez que suas realizações podem tornar-se antieconômicas dentro desse modelo de implementação. Por outro lado, arquiteturas paralelas, em que cada processador percorre uma seqüência distinta de movimentações, possibilitam a obtenção de implementações bastante eficientes para autômatos finitos não-determinísticos.

A seguir é mostrada a equivalência entre os autômatos finitos não-determinísticos e os determinísticos, no que diz respeito à classe de linguagens que eles são capazes de reconhecer. A **equivalência**, ou **equipotência**, de tais classes de autômato, constitui um dos mais importantes resultados da teoria dos autômatos finitos, sem paralelo para a maioria dos demais modelos de reconhecedores anteriormente mencionados, uma vez que garante ser sempre possível a aceitação de toda e qualquer linguagem regular através de um autômato determinístico.

Antes, porém, é preciso introduzir a **notação tabular** para a representação de autômatos finitos. De acordo com essa notação, cada linha da tabela representa um

estado distinto q do autômato, e cada coluna é associada a um elemento distinto σ de seu alfabeto de entrada. As células correspondentes à intersecção de cada linha com cada coluna são preenchidas com o elemento (conjunto) de 2^Q determinado por $\delta(q, \sigma)$.

Exemplo 3.11 Considere-se novamente o autômato finito não-determinístico M do Exemplo 3.10 cujo diagrama de estados é apresentado na Figura 3.5. A representação tabular de M é apresentada na Tabela 3.3.

	δ	a	b	c
$\rightarrow$	q_0	$\{q_1, q_2\}$		
$\leftarrow$	q_1		$\{q_1, q_2\}$	
$\leftarrow$	q_2			$\{q_2\}$

Tabela 3.3: Notação tabular para o autômato finito não-determinístico M da Figura 3.5

□

Na notação tabular, como se pode perceber, o estado inicial é indicado através do símbolo "$\rightarrow$", ao passo que os estados finais são indicados por "$\leftarrow$". O símbolo "$\leftrightarrow$" indica um estado que seja simultaneamente inicial e final.

Teorema 3.3 (Eliminação de não-determinismos) *Seja L a linguagem aceita por um autômato finito não-determinístico sem transições em vazio. Então é possível definir um autômato finito determinístico equivalente que aceita L.*

Justificativa A equivalência dos autômatos finitos não-determinísticos sem transições em vazio com os autômatos finitos determinísticos é mostrada através de um algoritmo que permite a construção de autômatos do segundo tipo a partir de autômatos do primeiro tipo, quaisquer que sejam eles.

O mecanismo de mapeamento é baseado na substituição de todas as transições não-determinísticas do autômato finito não-determinístico original por transições determinísticas para novos estados criados no novo autômato construído. A fim de garantir a equivalência das linguagens aceitas pelos dois autômatos, deve-se copiar, para cada um desses novos estados do autômato finito determinístico resultante, todas as transições que partem de estados que seriam atingidos pelas transições não-determinísticas do autômato finito não-determinístico original.

Em outras palavras, para cada transição não-determinística distinta presente no autômato original, o algoritmo cria um novo estado e a substitui por uma transição determinística para esse novo estado, copiando as transições dos estados que seriam atingidos pela realização da transição não-determinística. Como desse processo pode resultar a introdução de novos não-determinismos, torna-se necessário aplicá-lo de forma iterativa até a completa eliminação dos não-determinismos. A Figura 3.6 ilustra essas idéias.

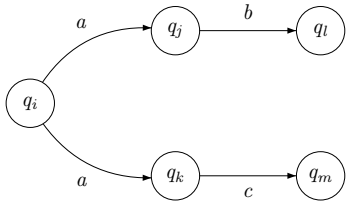

Figura 3.6: Situação não-determinística original

Suponha-se um autômato qualquer que apresente, como parte de sua especificação, uma transição não-determinística $\delta(q_i, a) = \{q_j, q_k\}$. Ao criar um novo estado $q_j\,q_k$ e substituir as transições anteriores por uma única e nova transição determinística $\delta(q_i, a) = q_j\,q_k$, consegue-se fazer com que o autômato modificado aceite a mesma linguagem que o original, sem no entanto apresentar comportamento não-determinístico. De fato, neste exemplo, em ambas as versões os estados atingidos pelas cadeias ab e ac são os mesmos: respectivamente q_l e q_m. A Figura 3.7 ilustra o autômato modificado.

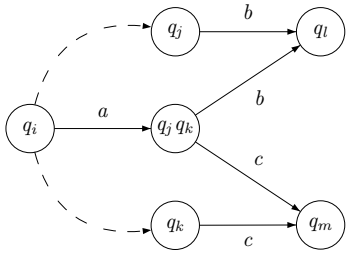

Figura 3.7: Situação determinística equivalente à da Figura 3.6

Fica claro, também, que se pelo menos um dos estados, q_j ou q_k, for um estado final, o mesmo deverá acontecer com o estado $q_j\,q_k$, já que em ambas as versões a cadeia a deve ser aceita pelo autômato. Por outro lado, caso haja coincidência entre os símbolos b e c, um novo não-determinismo será introduzido no estado $q_j\,q_k$. Daí a necessidade de se repetir o procedimento, removendo a cada iteração todos os novos não-determinismos que venham a ser introduzidos.

Seja $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ o autômato finito não-determinístico originalmente considerado e $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ o autômato finito determinístico correspondente que se deseja obter. A obtenção de M_2 a partir de M_1 pode ser efetuada através do Algoritmo 3.4.

Algoritmo 3.4 (Eliminação de não-determinismos) *Obtenção de um autômato finito determinístico M_2 a partir de um autômato finito não-determinístico M_1.*

- Entrada: um autômato não-determinístico $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, com $\delta_1 : Q_1 \times \Sigma \rightarrow 2^{Q_1}$;

- Saída: um autômato determinístico $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$, com $\delta_2 : Q_2 \times \Sigma \rightarrow Q_2$, tal que $L(M_2) = L(M_1)$;
- Método:

1. $Q_2 \leftarrow \varnothing$;
2. $\forall\, i \geq 0$, se $q_{1i} \in Q_1$ então $Q_2 \leftarrow Q_2 \cup \{q_{2i}\}$;
3. $F_2 \leftarrow \varnothing$;
4. $\forall\, i \geq 0$, se $q_{1i} \in F_1$ então $F_2 \leftarrow F_2 \cup \{q_{2i}\}$;
5. $\delta' \leftarrow \varnothing$;
6. $\forall\, q_{1i} \in Q_1, \sigma \in \Sigma$, se $\delta_1(q_{1i}, \sigma) = \{q_{11}, ..., q_{1n}\}, n \geq 1$ então $\delta_2(q_{2i}, \sigma) = \{q_{21}, ..., q_{2n}\}$;
7. Substituir todos os elementos $\{q_{2i}\}$ de δ_2 por q_{2i};
8. Enquanto houver transições não-determinísticas em δ_2, faça:

   a) Selecione uma transição não-determinística qualquer $\delta_2(q, \sigma)$, e considere $\delta_2(q, \sigma) = \{q_{21}, ..., q_{2i}, ..., q_{2n}\}, n \geq 2$;

   b) Acrescente um novo estado $q_{21}...q_{2i}...q_{2n}$ à tabela de transição de estados (nesta notação, os estados do conjunto são concatenados formando uma cadeia, em que os índices dos estados estão organizados em ordem crescente, ou em qualquer outra ordem conveniente); se $q_{2i} = q_{2i1}...q_{2im}$, considerar a ordenação de todos os estados obtidos pela substituição de q_{2i} por $q_{2i1}...q_{2im}$ em $q_{21}...q_{2i}...q_{2n}$;

   c) Substitua, na tabela, todas as referências a $\{q_{21}, ..., q_{2n}\}$ por $q_{21}...q_{2n}$;

   d) Para cada $\sigma \in \Sigma$, faça:

      i. $\delta_2(q_{21}...q_{2n}, \sigma) \leftarrow \varnothing$;

      ii. Para cada estado $q_{2j} \in \{q_{21}, ..., q_{2n}\}$, faça:

         A. $\delta_2(q_{21}...q_{2n}, \sigma) \leftarrow \delta_2(q_{21}...q_{2n}, \sigma) \cup \delta_2(q_{2j}, \sigma)$;

         B. Se $q_{2j} \in F_2$, então $F_2 \leftarrow F_2 \cup \{q_{21}...q_{2n}\}$.

■

Exemplo 3.12 Considere-se uma vez mais o autômato não-determinístico M do Exemplo 3.10, representado na Figura 3.5 e na Tabela 3.3. A aplicação do Algoritmo 3.4 a M conduz à obtenção do autômato da Tabela 3.4.

	δ'	a	b	c
$\rightarrow$	q_0	$q_1\,q_2$		
$\leftarrow$	q_1		$q_1\,q_2$	
$\leftarrow$	q_2			q_2
$\leftarrow$	$q_1\,q_2$		$q_1\,q_2$	q_2

Tabela 3.4: Autômato determinístico equivalente ao autômato M da Tabela 3.3

Observe-se, inicialmente, a criação de um novo estado, denominado $q_1\,q_2$, em decorrência da presença do não-determinismo $\{q_1, q_2\}$ na tabela de transição de estados original. As células desse novo estado, ou seja, suas transições, são preenchidas de acordo com o seguinte critério: como $\delta(q_0, a) = \{q_1, q_2\}$, deve-se considerar todas as transições que partem de q_1 e de q_2, respectivamente $\delta(q_1, b)$ e $\delta(q_2, c)$. Assim, o conteúdo de $\delta(q_1, b)$ é copiado para $\delta(q_1\,q_2, b)$ e o conteúdo de $\delta(q_2, c)$ é copiado para $\delta(q_1\,q_2, c)$. Finalmente, como $\delta(q_2, c)$ é um estado final, o novo estado $q_1\,q_2$ também torna-se final.

Note-se ainda que, apesar de o autômato inicial deste exemplo apresentar duas transições não-determinísticas ($\delta(q_0, a)$ e $\delta(q_1, b)$), apenas um novo estado foi adicionado ($q_1\,q_2$). Isso decorre do fato de que os estados de destino, em ambos os casos, são os mesmos, não havendo necessidade de se memorizar a forma através da qual tais estados foram atingidos.

Como se pode perceber, a eliminação de não-determinismos implica a criação de novos estados (Q'), altera a função de transição de estados (δ') e, eventualmente, acrescenta estados finais ao autômato resultante (F'). É comum que, após essa transformação, o autômato finito determinístico resultante contenha estados inacessíveis, ou seja, estados que não podem ser atingidos a partir do estado inicial por nenhum caminho.

A Figura 3.8 mostra o diagrama de estados do autômato determinístico obtido neste exemplo. Deve-se observar que o estado q_1 tornou-se inacessível como resultado da aplicação do método de eliminação de não-determinismos.

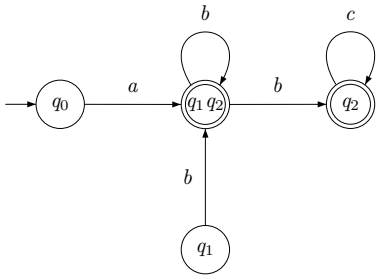

Figura 3.8: Autômato determinístico equivalente ao autômato M da Figura 3.5

□

Exemplo 3.13 Considere-se o autômato finito não-determinístico representado na Figura 3.9.

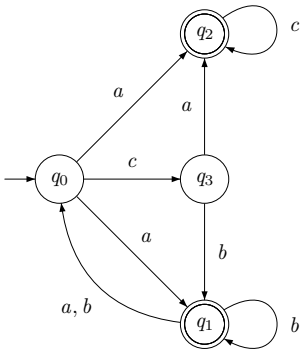

Figura 3.9: Autômato não-determinístico do Exemplo 3.13

A Tabela 3.5 representa o autômato da Figura 3.9.

	δ	a	b	c
$\rightarrow$	q_0	$\{q_1, q_2\}$		$\{q_3\}$
$\leftarrow$	q_1	$\{q_0\}$	$\{q_0, q_1\}$	
$\leftarrow$	q_2			$\{q_2\}$
	q_3	$\{q_2\}$	$\{q_1\}$	

Tabela 3.5: Eliminação de não-determinismos, autômato inicial

Os seguintes passos correspondem à aplicação do Algoritmo 3.4.

- Substituir $\{q_0\}$ por q_0, $\{q_1\}$ por q_1, $\{q_2\}$ por q_2 e $\{q_3\}$ por q_3:

	δ	a	b	c
$\rightarrow$	q_0	$\{q_1, q_2\}$		q_3
$\leftarrow$	q_1	q_0	$\{q_0, q_1\}$	
$\leftarrow$	q_2			q_2
	q_3	q_2	q_1	

Tabela 3.6: Eliminação de não-determinismos, passo 1

- Criar um novo estado $q_1\,q_2$, substituindo $\{q_1, q_2\}$ na tabela por $q_1\,q_2$.

	δ	a	b	c
$\rightarrow$	q_0	$q_1\,q_2$		q_3
$\leftarrow$	q_1	q_0	$\{q_0, q_1\}$	
$\leftarrow$	q_2			q_2
	q_3	q_2	q_1	
$\leftarrow$	$q_1\,q_2$	q_0	$\{q_0, q_1\}$	q_2

Tabela 3.7: Eliminação de não-determinismos, passo 2

- Criar um novo estado $q_0\,q_1$, substituindo $\{q_0, q_1\}$ na tabela por $q_0\,q_1$.

	δ	a	b	c
$\rightarrow$	q_0	$q_1\,q_2$		q_3
$\leftarrow$	q_1	q_0	$q_0\,q_1$	
$\leftarrow$	q_2			q_2
	q_3	q_2	q_1	
$\leftarrow$	$q_1\,q_2$	q_0	$q_0\,q_1$	q_2
$\leftarrow$	$q_0\,q_1$	$\{q_1\,q_2, q_0\}$	$q_0\,q_1$	q_3

Tabela 3.8: Eliminação de não-determinismos, passo 3

- Criar um novo estado $q_0\,q_1\,q_2$, substituindo $\{q_1\,q_2, q_0\}$ na tabela por $q_0\,q_1\,q_2$.

	δ	a	b	c
$\rightarrow$	q_0	$q_1\,q_2$		
$\leftarrow$	q_1	q_0	$q_0\,q_1$	
$\leftarrow$	q_2			q_2
	q_3	q_2	q_1	
$\leftarrow$	$q_1\,q_2$	q_0	$q_0\,q_1$	q_2
$\leftarrow$	$q_0\,q_1$	$q_0\,q_1\,q_2$	$q_0\,q_1$	
$\leftarrow$	$q_0\,q_1\,q_2$	$q_0\,q_1\,q_2$	$q_0\,q_1$	$\{q_2, q_3\}$

Tabela 3.9: Eliminação de não-determinismos, passo 4

- Criar um novo estado $q_2\,q_3$, substituindo $\{q_2, q_3\}$ na tabela por $q_2\,q_3$.

	δ	a	b	c
$\rightarrow$	q_0	$q_1\, q_2$		
$\leftarrow$	q_1	q_0	$q_0\, q_1$	
$\leftarrow$	q_2			q_2
	q_3	q_2	q_1	
$\leftarrow$	$q_1\, q_2$	q_0	$q_0\, q_1$	q_2
$\leftarrow$	$q_0\, q_1$	$q_0\, q_1\, q_2$	$q_0\, q_1$	
$\leftarrow$	$q_0\, q_1\, q_2$	$q_0\, q_1\, q_2$	$q_0\, q_1$	$q_2\, q_3$
$\leftarrow$	$q_2\, q_3$	q_2	q_1	q_2

Tabela 3.10: Eliminação de não-determinismos, autômato final

A Figura 3.10 apresenta o diagrama de estados do autômato determinístico obtido. O estado q_3, que é inacessível, não está mostrado na figura.

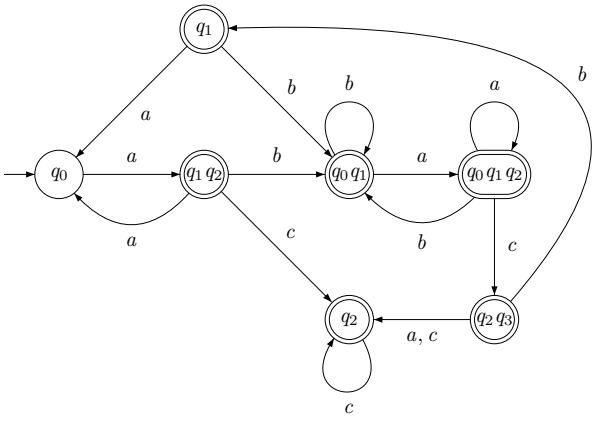

Figura 3.10: Autômato determinístico equivalente ao da Figura 3.9

□

Os estados criados de acordo com o Algoritmo 3.4 têm como função reproduzir o comportamento do autômato não-determinístico original no novo autômato determinístico, considerando-se todas as transições possíveis de serem executadas em cada um dos estados que são destinos de uma mesma transição não-determinística do autômato original.

Se, por exemplo, $\delta(q_i, \sigma) = \{q_j, q_k, q_m\}$, o estado $q_j\, q_k\, q_m$ do autômato determinístico atenderá ao propósito de permitir que o novo autômato se movimente, a partir deste estado, com transições similares às originalmente presentes em cada um dos estados q_j, q_k e q_m do autômato não-determinístico.

Dessa forma, o surgimento de novos estados no autômato determinístico é limitado pela quantidade de combinações distintas que podem ser feitas entre os estados do autômato não-determinístico original. Se $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ é o autômato original

não-determinístico, e $M_2 = (Q_2, \Sigma, \delta_2, q_0, F_2)$ é o autômato determinístico equivalente, então $\left|Q_2\right| \leq 2^{\left|Q_1\right|} - 1$, uma vez que toda combinação de estados deverá conter pelo menos um estado do autômato original.

Exemplo 3.14 A eliminação dos não-determinismos do autômato finito representado na Tabela 3.11 resulta no autômato determinístico da Tabela 3.12.

Como se pode perceber, $Q_1 = \{q_0, q_1, q_2\}$ e $Q_2 = \{q_0, q_1, q_2, q_0\,q_1, q_0\,q_2, q_1\,q_2, q_0\,q_1\,q_2\}$. Além disso, $\left|Q_1\right| = 3$ e $\left|Q_2\right| = 2^3 - 1 = 7$. Trata-se, portanto, de um autômato determinístico no qual todas as combinações possíveis dos estados do autômato original não-determinístico estão consideradas.

	δ	a	b
$\rightarrow$	q_0	$\{q_1, q_2\}$	
$\leftarrow$	q_1		
	q_2	$\{q_0, q_2\}$	$\{q_0, q_1\}$

Tabela 3.11: Autômato finito não-determinístico do Exemplo 3.14

	δ	a	b
$\rightarrow$	q_0	$q_1\,q_2$	
$\leftarrow$	q_1		
	q_2	$q_0\,q_2$	$q_0\,q_1$
$\leftarrow$	$q_1\,q_2$	$q_0\,q_2$	$q_0\,q_1$
$\leftarrow$	$q_0\,q_1$	$q_1\,q_2$	
	$q_0\,q_2$	$q_1\,q_2, q_0\,q_2$	$q_0\,q_1$
$\leftarrow$	$q_0\,q_1\,q_2$	$q_0\,q_1\,q_2$	$q_0\,q_1$

Tabela 3.12: Autômato finito determinístico equivalente ao da Tabela 3.11

□

Note-se que nem sempre a eliminação de não-determinismos de um autômato finito faz surgir tantos novos estados, como acontece no Exemplo 3.14. Na prática, costumam surgir alguns estados novos, mas não todos os possíveis, e também alguns dos estados antigos eventualmente tornam-se inacessíveis, podendo ser eliminados, de forma que, tipicamente, há um aumento do número de estados, mas em geral esse número não costuma atingir esse limite extremo.

Como conclusão da apresentação do Teorema 3.3, e com base no Algoritmo 3.4, deve-se acrescentar que é possível garantir, no caso geral, a existência de um autômato finito determinístico equivalente a qualquer autômato finito não-determinístico fornecido.

Dessa maneira, o fato de um autômato finito ser não-determinístico não o torna mais poderoso quanto à classe de linguagens que é capaz de reconhecer, quando comparado com os autômatos finitos determinísticos. Por se tratar, este último, de um modelo de reconhecimento que permite gerar implementações extremamente eficientes, conclui-se ser sempre possível a obtenção de modelos com tais características, independentemente da forma como o autômato se manifesta originalmente quanto ao seu determinismo.

Por outro lado, a existência de autômatos não-determinísticos que sejam equivalentes a autômatos determinísticos é imediata, pois a incorporação de não-determinismos

pode ser feita trivialmente, sem alterar a linguagem aceita pelo autômato: basta, por exemplo, incorporar um caminho adicional alternativo que aceite qualquer seqüência de símbolos, a partir de qualquer estado, iniciada por um símbolo que já seja consumido a partir daquele estado, sem, no entanto, permitir que alguma configuração final seja atingida.

Exemplo 3.15 Considere-se M, o autômato determinístico da Figura 3.11:

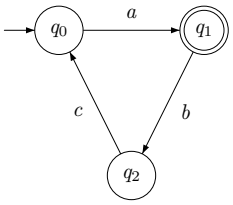

Figura 3.11: Autômato M determinístico que aceita $a(bca)^*$

O acréscimo de um único novo estado q_3 e da transição $\delta(q_2, c) = q_3$ já seria suficiente para tornar M não-determinístico, sem no entanto alterar a linguagem por ele aceita ($a(bca)^*$). Naturalmente, inúmeros autômatos podem ser construídos dessa maneira. A Figura 3.12 apresenta um exemplo.

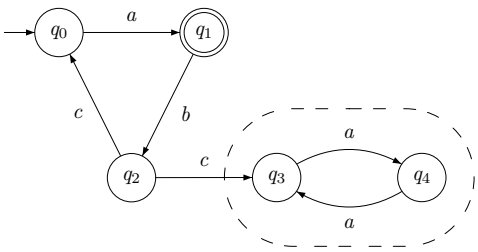

Figura 3.12: Autômato não-determinístico equivalente ao autômato da Figura 3.11

□

Autômatos Finitos Não-Determinísticos com Transições em Vazio

Autômatos finitos não-determinísticos que apresentam **transições em vazio** são aqueles que admitem transições de um estado para outro com ϵ, além das transições normais, que utilizam os símbolos do alfabeto de entrada. Transições em vazio podem ser executadas sem que seja necessário consultar o símbolo corrente da fita de entrada, e sua execução nem sequer causa o deslocamento do cursor de leitura. Com a introdução de transições em vazio, a função de transição para autômatos finitos não-determinísticos passa a ter seu domínio alterado para:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$

Quando um autômato transita em vazio, isso significa que ele muda de estado sem consultar a cadeia de entrada. Sempre que ocorrer a coexistência entre alguma transição em vazio e outras transições (vazias ou não) com origem em um mesmo estado, isso

acarreta a necessidade de uma escolha arbitrária da transição a ser aplicada na respectiva configuração, e isso, por sua vez, caracteriza a manifestação de um não-determinismo.

Exemplo 3.16 Seja $M = (Q, \Sigma, \delta, q_0, F)$ um autômato finito com transições em vazio.

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{a, b\} \\ \delta &= \{(q_0, a) \rightarrow \{q_0\}, (q_0, \epsilon) \rightarrow \{q_1\}, (q_1, b) \rightarrow \{q_1\}\} \\ F &= \{q_1\} \end{aligned}$$

A linguagem aceita por esse autômato é a^*b^*, conforme pode ser deduzido a partir do diagrama de estados da Figura 3.13.

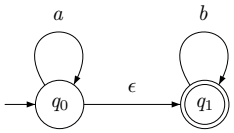

Figura 3.13: Autômato com transição em vazio

Tomando-se a cadeia de entrada ab como exemplo, as duas seqüências de movimentação possíveis a partir da configuração inicial seriam:

1. $(q_0, ab) \vdash (q_0, b) \vdash (q_1, b) \vdash (q_1, \epsilon)$ (sucesso)
2. $(q_0, ab) \vdash (q_1, ab)$ (impasse)

No segundo caso, o impasse ocorre devido à aplicação da transição $(q_0, \epsilon) \rightarrow q_1$ logo ao início do reconhecimento, antes de ser efetuada a leitura do símbolo a. No primeiro caso, ocorre a aceitação da sentença, pois a utilização da transição em vazio foi efetuada em um ponto favorável pelo autômato, ou seja, entre a utilização dos símbolos a e b da cadeia de entrada. □

Apesar de constituir uma generalização da definição dos autômatos finitos não-determinísticos (basta comparar as respectivas funções de transição: $Q \times \Sigma \rightarrow 2^Q$ *versus* $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$), é possível demonstrar que a incorporação de transições em vazio ao autômato em nada aumenta o seu poder de reconhecimento. Em outras palavras, toda e qualquer linguagem que seja aceita por um autômato finito não-determinístico que apresente transições em vazio pode também ser aceita por um autômato equivalente, sem transições em vazio.

A equivalência dessas duas classes de autômatos é desenvolvida a partir da discussão de um algoritmo que possibilite a conversão sistemática de autômatos finitos quaisquer em uma versão equivalente, porém isenta de transições em vazio.

Teorema 3.4 (Eliminação de transições em vazio, versão 1) *Todo autômato que contenha transições em vazio gera uma linguagem que é aceita por algum autômato finito que não contém transições em vazio.*

Justificativa Sejam $M = (Q, \Sigma, \delta, q_0, F)$ um autômato finito contendo transições em vazio, representado em notação tabular, e $N = (Q, \Sigma, \delta', q_0, F')$ o autômato finito sem transições em vazio correspondente que a partir dele se deseja obter. A obtenção de N a partir de M pode ser efetuada através do Algoritmo 3.5.

Algoritmo 3.5 (Eliminação de transições em vazio, versão 1) *Obtenção de um autômato finito N, sem transições em vazio, a partir de um autômato finito M, com transições em vazio.*

- Entrada: um autômato finito com transições em vazio M;
- Saída: um autômato finito sem transições em vazio N, tal que $L(N) = L(M)$;
- Método:

  1. Eliminação das transições em vazio

     Considere-se um estado qualquer $q_i \in Q$. Se houver uma transição em vazio de q_i para q_j, deve-se eliminá-la, copiando-se para a linha que representa o estado q_i todas as transições que partem dos estados q_j para os quais é feita a transição em vazio.

     Esse procedimento corresponde, em notação tabular, à realização de uma fusão ("merge") entre a linha do estado q_i que contém a transição em vazio para o estado-destino q_j e a própria linha do estado q_j, armazenando-se o resultado novamente na linha correspondente ao estado q_i.

     Havendo mais de uma transição em vazio indicadas, deve-se repetir cumulativamente o procedimento para todas elas.

     Se $\delta(q_i, \epsilon) \in F$, então $F' \leftarrow F' \cup \{q_i\}$, sendo que inicialmente $F' \leftarrow F$.

  2. Iteração

     Repetir o passo anterior para os demais estados do autômato, até que todos eles tenham sido considerados (ou seja, até que a última linha tenha sido atingida).

     Nos casos em que houver transições em vazio para estados que por sua vez também transitam em vazio para outros estados, será necessário iterar o procedimento várias vezes sobre a tabela, até que todas as transições em vazio tenham sido eliminadas.

■

O algoritmo funciona através da substituição de transições em vazio por cópias das transições que partem dos estados que seriam atingidos pelas transições em vazio. As únicas mudanças do autômato dizem respeito à função de transição δ', que se torna definida para alguns elementos do domínio de δ anteriormente indefinidos (δ', na notação tabular, torna-se menos esparsa), e ao conjunto de estados finais F', que eventualmente se amplia se existirem caminhos formados exclusivamente por transições em vazio que interligam estados finais a outros estados do autômato.

Exemplo 3.17 Considere-se o autômato finito M do Exemplo 3.16, representado na Tabela 3.13.

	δ	a	b	ϵ
$\rightarrow$	q_0	q_0		q_1
$\leftarrow$	q_1		q_1	

Tabela 3.13: Autômato original apresentando transições em vazio

Como há uma transição em vazio de q_0 para q_1, deve-se copiar as transições de q_1 para q_0 ($\delta(q_1, b)$ apenas, neste caso) e, além disso, considerar q_0 como estado final, uma vez que q_1 é estado final. A Tabela 3.14 representa a função de transição δ'.

	δ'	a	b
$\leftrightarrow$	q_0	q_0	q_1
$\leftarrow$	q_1		q_1

Tabela 3.14: Autômato equivalente ao da Tabela 3.13, porém isento de transições em vazio

O diagrama de estados do autômato finito correspondente, sem transições em vazio, é apresentado na Figura 3.14.

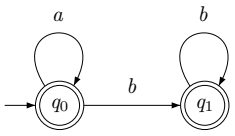

Figura 3.14: Autômato equivalente ao da Tabela 3.14, eliminadas as transições em vazio

□

A seguir, será apresentado um algoritmo alternativo para a eliminação de transições em vazio da especificação de um autômato finito. Diferentemente do anterior, este algoritmo não exige a manipulação da tabela que representa o autômato. É possível demonstrar que, apesar de não serem obrigatoriamente iguais, os autômatos obtidos pela aplicação dos dois métodos são equivalentes (no sentido de reconhecerem exatamente a mesma linguagem).

Inicialmente, são apresentadas duas novas definições: o Fechamento-ϵ de um estado e a função de transição estendida Δ.

A função **Fechamento-ϵ**: $Q \rightarrow 2^Q$ é definida de forma indutiva e pode ser calculada conforme o Algoritmo 3.6.

Algoritmo 3.6 (Fechamento-ϵ) *Cálculo do Fechamento-ϵ de um estado q_i.*

- Entrada: um autômato finito $M = (Q, \Sigma, \delta, Q_0, F)$ e um estado $q_i \in Q$;
- Saída: Fechamento-$\epsilon(q_i)$;
- Método:

  1. Fechamento-$\epsilon(q_i) \leftarrow \{q_i\}$;

2. Fechamento-$\epsilon(q_i) \leftarrow$ Fechamento-$\epsilon(q_i) \cup \delta(q_i, \epsilon)$;
3. $\forall\, q_j \in \delta(q_i, \epsilon)$, Fechamento-$\epsilon(q_i) \leftarrow$ Fechamento-$\epsilon(q_i) \cup$ Fechamento-$\epsilon(q_j)$.

A função Fechamento-$\epsilon(q_i)$ corresponde ao conjunto de estados que podem ser atingidos a partir do estado q_i pela aplicação exclusiva de transições em vazio.

Exemplo 3.18 Considere-se o autômato da Figura 3.15:

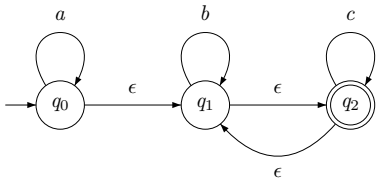

Figura 3.15: Autômato com transições em vazio

De acordo com a definição:

- Fechamento-$\epsilon(q_0) = \{q_0, q_1, q_2\}$
- Fechamento-$\epsilon(q_1) = \{q_1, q_2\}$
- Fechamento-$\epsilon(q_2) = \{q_1, q_2\}$

□

A **função de transição estendida** Δ: $Q \times \Sigma^* \rightarrow 2^Q$ também é definida de forma indutiva, conforme o Algoritmo 3.7.

Algoritmo 3.7 (Função de transição estendida Δ) *Cálculo da função de transição estendida* Δ.

- Entrada: um autômato finito $M = (Q, \Sigma, \delta, Q_0, F)$, um estado $q_i \in Q$ e uma cadeia $\alpha \in \Sigma^*$;
- Saída: $\Delta(q_i, \alpha)$;
- Método:
  1. $\Delta(q_i, \epsilon) \leftarrow$ Fechamento-$\epsilon(q_i)$;
  2. $\Delta(q_i, \gamma\sigma) \leftarrow \bigcup_k$ Fechamento-$\epsilon(q_k)$, com $q_j \in \Delta(q_i, \gamma), q_k \in \delta(q_j, \sigma), \gamma \in \Sigma^*, \sigma \in \Sigma$.

Exemplo 3.19 Considerando-se o autômato do Exemplo 3.18:

- O cálculo de $\Delta(q_0, a)$ corresponde a:
  $\Delta(q_0, \epsilon) =$ Fechamento-$\epsilon(q_0) = \{q_0, q_1, q_2\}$

$\delta(q_0, a) = \{q_0\}$
$\delta(q_1, a) = \varnothing$
$\delta(q_2, a) = \varnothing$
Fechamento-$\epsilon(q_0) \cup \varnothing \cup \varnothing$ = Fechamento-$\epsilon(q_0) = \{q_0, q_1, q_2\}$
Logo, $\Delta(q_0, a) = \{q_0, q_1, q_2\}$.

- O cálculo de $\Delta(q_0, ab)$ corresponde a:
  $\Delta(q_0, \epsilon)$ = Fechamento-$\epsilon(q_0) = \{q_0, q_1, q_2\}$
  $\delta(q_0, a) = \{q_0\}$
  $\delta(q_1, a) = \varnothing$
  $\delta(q_2, a) = \varnothing$
  Fechamento-$\epsilon(q_0) \cup \varnothing \cup \varnothing = \{q_0, q_1, q_2\}$
  $\delta(q_0, b) = \varnothing$
  $\delta(q_1, b) = \{q_1\}$
  $\delta(q_2, b) = \varnothing$
  $\varnothing \cup$ Fechamento-$\epsilon(q_1) \cup \varnothing = \{q_1, q_2\}$
  Logo, $\Delta(q_0, ab) = \{q_1, q_2\}$.

□

Considere-se $\alpha \in \Sigma^*$. Se $\alpha = \epsilon$, então $\Delta(q_i, \alpha)$ corresponde ao conjunto dos estados que podem ser alcançados a partir de q_i pelo uso exclusivo de transições em vazio. Se, no entanto, $\alpha \neq \epsilon$, $\Delta(q_i, \alpha)$ deve ser interpretado como sendo o conjunto dos estados que podem ser atingidos a partir do estado q_i, considerando-se todas as possibilidades de movimentação em vazio que o autômato oferece. Note-se que todos os símbolos de α deverão ser consumidos nessa operação.

Um caso particular da função Δ é especialmente interessante: aquele em que a cadeia α contém um único símbolo σ: $\Delta(q_i, \sigma), q_i \in Q, \sigma \in \Sigma$. Neste caso, $\Delta(q_i, \sigma)$ corresponde ao conjunto dos estados que podem ser atingidos após o consumo de σ, considerando-se todas as possibilidades de transições em vazio que podem ocorrer antes e depois do consumo do mesmo: ϵ ... $\epsilon\sigma$, $\sigma\epsilon$ ... ϵ, $\epsilon...\epsilon\sigma\epsilon...\epsilon$ ou simplesmente σ. Esse aspecto pode ser melhor compreendido com o auxílio da Figura 3.16, que apresenta a especificação parcial de um autômato contendo transições em vazio:

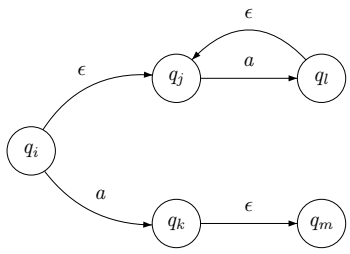

Figura 3.16: Autômato original, com transições em vazio

As seguintes relações são verdadeiras:

- $\Delta(q_i, a) = \{q_j, q_l, q_k, q_m\}$, pois:

  - $(q_i, a) \Rightarrow (q_j, a) \Rightarrow (q_l, \epsilon)$
  - $(q_i, a) \Rightarrow (q_j, a) \Rightarrow (q_l, \epsilon) \Rightarrow (q_j, \epsilon)$
  - $(q_i, a) \Rightarrow (q_k, \epsilon)$
  - $(q_i, a) \Rightarrow (q_k, \epsilon) \Rightarrow (q_m, \epsilon)$

- $\Delta(q_j, a) = \{q_j, q_l\}$, pois:
  - $(q_j, a) \Rightarrow (q_l, \epsilon)$
  - $(q_j, a) \Rightarrow (q_l, \epsilon) \Rightarrow (q_j, \epsilon)$

- $\Delta(q_l, a) = \{q_j, q_l\}$, pois:
  - $(q_l, a) \Rightarrow (q_j, a) \Rightarrow (q_l, \epsilon)$
  - $(q_l, a) \Rightarrow (q_j, a) \Rightarrow (q_l, \epsilon) \Rightarrow (q_j, \epsilon)$

- $\Delta(q_k, a) = \varnothing$
- $\Delta(q_m, a) = \varnothing$

A Figura 3.17 apresenta uma versão modificada desse mesmo trecho do autômato, tendo a função de transição original δ sido substituída pela nova função de transição Δ, acima calculada:

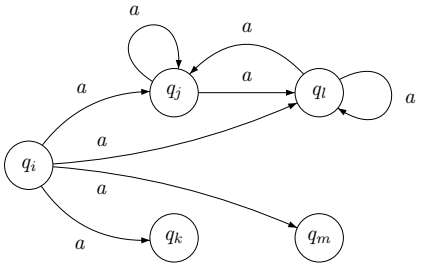

Figura 3.17: Autômato equivalente ao da Figura 3.16, eliminadas todas as transições em vazio

Exemplo 3.20 Dando seqüência ao Exemplo 3.19, apresenta-se a seguir o valor da função Δ para cada elemento do conjunto $Q \times \Sigma$. Em cada caso, são indicadas as várias possibilidades de movimentação que o autômato possui a partir de $q \in Q$ no reconhecimento da cadeia $\sigma \in \Sigma$, e que são consideradas no cálculo da função Δ:

- $\Delta(q_0, a) = \{q_0, q_1, q_2\}$
  - $(q_0, a) \vdash (q_0, \epsilon)$

  - $(q_0, a) \vdash (q_0, \epsilon) \vdash (q_1, \epsilon)$
  - $(q_0, a) \vdash (q_0, \epsilon) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - $(q_0, a) \vdash (q_0, \epsilon) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon)$
  - $(q_0, a) \vdash (q_0, \epsilon) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - etc.

- $\Delta(q_1, a) = \varnothing$
  - O símbolo a não pode ser consumido a partir do estado q_1
- $\Delta(q_2, a) = \varnothing$
  - O símbolo a não pode ser consumido a partir do estado q_2
- $\Delta(q_0, b) = \{q_1, q_2\}$
  - $(q_0, b) \vdash (q_1, b) \vdash (q_1, \epsilon)$
  - $(q_0, b) \vdash (q_1, b) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - $(q_0, b) \vdash (q_1, b) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon)$
  - etc.
- $\Delta(q_1, b) = \{q_1, q_2\}$
  - $(q_1, b) \vdash (q_1, \epsilon)$
  - $(q_1, b) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - $(q_1, b) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon)$
  - $(q_1, b) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - etc.
- $\Delta(q_2, b) = \{q_1, q_2\}$
  - $(q_2, b) \vdash (q_1, b) \vdash (q_1, \epsilon)$
  - $(q_2, b) \vdash (q_1, b) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - $(q_2, b) \vdash (q_1, b) \vdash (q_2, b) \vdash (q_1, b) \vdash (q_1, \epsilon)$
  - $(q_2, b) \vdash (q_1, b) \vdash (q_2, b) \vdash (q_1, b) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - etc.
- $\Delta(q_0, c) = \{q_1, q_2\}$
  - $(q_0, c) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon)$
  - $(q_0, c) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon)$

  - $(q_0, c) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - etc.

- $\Delta(q_1, c) = \{q_1, q_2\}$
  - $(q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon)$
  - $(q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon)$
  - $(q_1, c) \vdash (q_2, c) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon)$
  - $(q_1, c) \vdash (q_2, c) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon)$
  - etc.

- $\Delta(q_2, c) = \{q_1, q_2\}$
  - $(q_2, c) \vdash (q_2, \epsilon)$
  - $(q_2, c) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon)$
  - $(q_2, c) \vdash (q_2, \epsilon) \vdash (q_1, \epsilon) \vdash (q_2, \epsilon)$
  - $(q_2, c) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon)$
  - $(q_2, c) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon)$
  - etc.

□

A linguagem L aceita por um autômato finito M definido através da função de transição Δ pode ser, portanto, representada também como:

$$L(M) = \{w \in \Sigma^* \mid \Delta(q_0, w) \cap F \neq \varnothing\}$$

O Teorema 3.4, que estabelece a equivalência dos autômatos finitos com e sem transições em vazio, está demonstrado novamente a seguir, usando desta vez como argumento o Algoritmo 3.8, que sintetiza a segunda técnica estudada. O teorema complementar, acerca da existência de autômatos com transições em vazio que sejam equivalentes a autômatos sem transições em vazio, dispensa demonstração, pois a adição de transições em vazio pode ser feita com facilidade, sem alterar a linguagem aceita pelo autômato.

Teorema 3.5 (Eliminação de transições em vazio, versão 2) *Todo autômato que contenha transições em vazio gera uma linguagem que é aceita por algum autômato finito que não contém transições em vazio.*

Justificativa Sejam $M_1 = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$ um autômato finito contendo transições em vazio e Δ_1 a função de transição estendida que o caracteriza. O autômato $M_2 = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$, construído de acordo com os passos do Algoritmo 3.8, é tal que $L(M_2) = L(M_1)$.

Algoritmo 3.8 (Eliminação de transições em vazio, versão 2) *Eliminação de transições em vazio usando a função de transição estendida Δ.*

- Entrada: um autômato finito $M_1 = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$, com transições em vazio;

- Saída: um autômato finito $M_2 = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$, sem transições em vazio, e tal que $L(M_2) = L(M_1)$;
- Método:

  1. $Q_2 \leftarrow \varnothing$;
  2. $\forall\, i \geq 0$, se $q_{1i} \in Q_1$ então $Q_2 \leftarrow Q_2 \cup \{q_{2i}\}$;
  3. $F_2 \leftarrow \varnothing$;
  4. $\forall\, i \geq 0$, se $q_{1i} \in F_1$ então $F_2 \leftarrow F_2 \cup \{q_{2i}\}$;
  5. Se Fechamento-$\epsilon(q_{10}) \cap F_1 \neq \varnothing$, então $F_2 \leftarrow F_2 \cup \{q_{20}\}$;
  6. $\delta_2 \leftarrow \varnothing$;
  7. $\forall\, q_{1i} \in Q_1, \sigma_m \in \Sigma$, se $\Delta_1(q_{1i}, \sigma_m) = \{q_{1m}, q_{1n}, ..., q_{1p}\}$, então $\delta_2 \leftarrow \delta_2 \cup \{(q_{2i}, \sigma_m) \rightarrow \{q_{2m}, q_{2n}, ..., q_{2p}\}\}$.

A definição da função δ_2 a partir da função Δ_1 faz com que as transições em vazio de M_1 sejam substituídas por transições não-vazias em M_2, de tal forma que todos os estados possíveis de serem atingidos pela ação de transições que consumam algum símbolo em M_1 sejam os mesmos quando as transições ocorrem em M_2. Assim, M_2 simula M_1.

Por outro lado, a eliminação das transições em vazio de M_1 pode provocar um efeito colateral indesejado em M_2: caso a cadeia vazia pertença à linguagem aceita por M_1, a mesma não será aceita por M_2. Se a cadeia vazia é aceita por M_1, então uma das duas situações é verdadeira:

- q_{01} é estado final, ou
- Fechamento-$\epsilon(q_{01}) \cap F_1 \neq \varnothing$

Se $q_{01} \in F_1$, então $q_{02} \in F_2$ e $\epsilon \in L(M_2)$, logo $L(M_1) = L(M_2)$. Se, no entanto, $q_{01} \notin F_1$, então q_{02} deverá ser acrescentado ao conjunto de estados finais de M_2, a fim de que a cadeia vazia possa ser aceita por M_2 também. Se $\epsilon \notin L(M_1)$, não há nada a fazer. Em outras palavras: se houver um caminho formado apenas por transições vazias entre q_{01} e algum estado final de M_1, q_{02} deverá ser tornado também um estado final em M_2■

Exemplo 3.21 A eliminação de transições em vazio no autômato do Exemplo 3.20 dá origem ao autômato da Tabela 3.15.

	δ_2	a	b	c
$\leftrightarrow$	q_{20}	$\{q_{20}, q_{21}, q_{22}\}$	$\{q_{21}, q_{22}\}$	$\{q_{21}, q_{22}\}$
	q_{21}	$\varnothing$	$\{q_{21}, q_{22}\}$	$\{q_{21}, q_{22}\}$
$\leftarrow$	q_{22}	$\varnothing$	$\{q_{21}, q_{22}\}$	$\{q_{21}, q_{22}\}$

Tabela 3.15: Autômato equivalente ao do Exemplo 3.20, eliminadas as suas transições em vazio

Como $q_{12} \in$ Fechamento-$\epsilon(q_{10})$, então q_{20} torna-se um estado final no novo autômato cujas transições em vazio foram eliminadas. A Figura 3.18 representa o autômato da Tabela 3.15 na forma de um diagrama de estados.

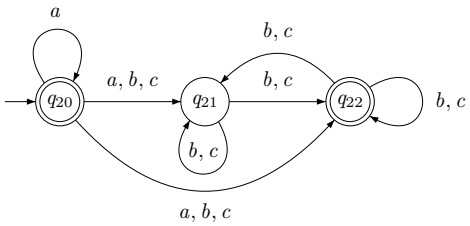

Figura 3.18: Autômato equivalente ao do Exemplo 3.20, eliminadas as suas transições em vazio

□

Os modelos de autômato finito considerados até o presente momento foram:

1. Determinístico sem transições em vazio, com $\delta : Q \times \Sigma \rightarrow Q$;
2. Não-determinístico sem transições em vazio, com $\delta : Q \times \Sigma \rightarrow 2^Q$;
3. Não-determinístico com transições em vazio, com $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$;

Para completar o quadro de possibilidades, define-se o modelo de autômato finito determinístico com transições em vazio:

4. Determinístico com transições em vazio, com $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow Q$.

Considerados todos os casos, cumpre retomar a discussão sobre o conceito de "não-determinismo" e a relação do mesmo com a forma através da qual são definidas as funções de transição dos autômatos finitos.

Se, por um lado, os autômatos dos modelos (2) e (3) são ditos não-determinísticos, isso não significa que, em casos particulares, sua operação não possa ocorrer de forma determinística (ver Exemplo 3.22). Por outro lado, os autômatos do modelo (4), apesar de denominados determinísticos, podem perfeitamente exibir um comportamente não-determinístico durante sua operação (ver Exemplo 3.23). Os autômatos do modelo (1), por sua vez, exibem sempre um comportamento determinístico.

Exemplo 3.22 Seja $M_1 = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$ um autômato finito não-determinístico cuja função de transição δ é definida como:

$$\begin{aligned} \delta(q_0, a) &= \{q_0\} \\ \delta(q_0, b) &= \{q_1\} \\ \delta(q_1, \epsilon) &= \{q_2\} \end{aligned}$$

Qualquer que seja a configuração corrente (q_i, α) considerada, existe sempre, no máximo, uma única transição de M_1 que pode ser aplicada e, portanto, no máximo, uma única próxima configuração possível. Logo, a operação de M_1 é sempre determinística. □

Exemplo 3.23 Seja $M_2 = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_1\})$ um autômato finito não-determinístico cuja função de transição δ é definida como:

$$\begin{aligned} \delta(q_0, a) &= q_0 \\ \delta(q_0, b) &= q_0 \\ \delta(q_0, \epsilon) &= q_1 \end{aligned}$$

Considere-se a cadeia de entrada a. As seguintes seqüências de movimentações são válidas em M_2:

- $(q_0, a) \vdash (q_0, \epsilon)$
- $(q_0, a) \vdash (q_1, a)$

Logo, a operação de M_2 é, nesse caso, não-determinística. □

Como mostram os Exemplos 3.22 e 3.23, o que efetivamente determina o comportamento que um autômato exibe durante a sua operação (se determinístico ou não) não é o formato genérico da função de transição adotoda, mas as especificidades de sua própria definição.

Assim, um autômato não-determinístico do modelo (2) pode operar de forma determinística se:

- $\nexists q \in Q, \sigma \in \Sigma$, tal que $\left|\delta(q, \sigma)\right| \geq 2$;

Um autômato não-determinístico do modelo (3) pode operar de forma determinística se:

- $\nexists q \in Q, \sigma \in \Sigma$, tal que $\left|\delta(q, \sigma)\right| \geq 2$, e
- $\nexists q \in Q$, tal que $\left|\delta(q, \epsilon)\right| \geq 2$, e
- $\nexists q \in Q, \sigma \in \Sigma$, tal que $\left|\delta(q, \sigma)\right| \geq 1$ e $\left|\delta(q, \epsilon)\right| \geq 1$.

Um autômato determinístico do modelo (4), por sua vez, pode operar de forma não-determinística, se:

- $\exists\, q \in Q, \sigma \in \Sigma$ tal que $\delta(q, \sigma)$ e $\delta(q, \epsilon)$ são definidas.

Portanto, o (não-)determinismo de um autômato transcende o formato geral de sua função de transição δ, dependendo, efetivamente, das características específicas dessa mesma função. A única exceção são os autômatos do modelo (1), cuja operação é sempre determinística, independentemente de como seja definida a função de transição.

Estados Inacessíveis e Inúteis

Nem todos os estados de um autômato necessariamente contribuem para a definição da linguagem por ele aceita. Estados inacessíveis e estados inúteis são os mais importantes representantes desta categoria de estados, os primeiros porque não podem ser alcançados a partir do estado inicial do autômato, e os demais porque não levam a nenhum dos estados finais. Na Figura 3.19, q_1 é um estado inacessível e q_2 um estado inútil.

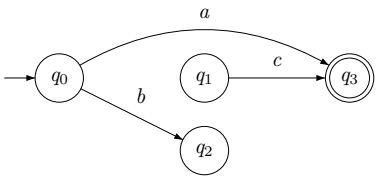

Figura 3.19: Ilustração dos conceitos de estado inacessível (q_1) e de estado inútil (q_2)

Por se tratar de estados sem relevância, no que se refere à linguagem que se está definindo, é geralmente desejável que os mesmos sejam simplesmente eliminados do autômato, possibilitando assim sua simplificação.

Estados inacessíveis são aqueles para os quais não existe no autômato qualquer caminho, formado por transições válidas, que permita atingi-los a partir do estado inicial do autômato. Eles podem ocorrer, por exemplo, como conseqüência direta da aplicação dos métodos anteriormente propostos para a eliminação de não-determinismos e/ou da remoção de transições em vazio. O método a seguir apresentado permite a identificação e a eliminação sistemática de estados inacessíveis eventualmente presentes em um autômato M qualquer. Observe-se que o estado inicial de um autômato é sempre acessível, fato este que será explorado como base do raciocínio indutivo implícito no algoritmo proposto.

Seja $M = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$ um autômato finito qualquer. Formalmente, um estado $q_{1i} \in Q_1$ é dito "inacessível" quando não existir caminho, formado por transições válidas, que conduza o autômato do seu estado inicial q_{10} até o estado q_{1i}. Em outras palavras, não existe $\alpha \in \Sigma^*$ tal que $\delta(q_{10}, \alpha) = q_{1i}$. Caso contrário, o estado q_{1i} é dito "acessível".

Estados inacessíveis não contribuem para a definição da linguagem aceita por M, podendo ser sistematicamente identificados e eliminados do conjunto de estados, sem prejuízo para a linguagem aceita pelo autômato. O Algoritmo 3.9, a seguir esboçado, permite construir um autômato finito $N = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$ isento de estados inacessíveis, tal que $L(N) = L(M)$.

Algoritmo 3.9 (Eliminação de estados inacessíveis, versão 1) *Obtenção de um autômato sem estados inacessíveis equivalente a outro com estados inacessíveis.*

- Entrada: um autômato finito $M = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$;
- Saída: um autômato finito $N = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$, isento de estados inacessíveis, e tal que $L(N) = L(M)$;
- Método:

  1. $Q_{20} \leftarrow \{q_{20}\}$;
  2. $i \leftarrow 1$;
  3. $Q_{2i} \leftarrow Q_{2(i-1)} \cup \{q_{2k} \mid q_{1k} \in Q_1 \text{ e } \delta_1(p, \sigma) = q_{1k}, p \in Q_{2(i-1)}, \sigma \in (\Sigma \cup \{\epsilon\})\}$;
  4. Se $Q_{2i} \neq Q_{2(i-1)}$, então $i \leftarrow i + 1$ e desviar para (3); caso contrário, $Q_2 \leftarrow Q_{2i}$;

5. $F_2 \leftarrow \{q_{2k} \in Q_2 \mid q_{1k} \in F_1\}$;

6. $\delta_2 \leftarrow \{(q_{2i}, \sigma) \rightarrow q_{2j} \mid (q_{1i}, \sigma) \rightarrow q_{1j} \in \delta_1 \text{ e } q_{2i}, q_{2j} \in Q_2\}$.

Considere-se a função de transição δ representada na forma tabular. Acrescentem-se duas novas colunas à tabela: a primeira, denominada "acessível", com a função de marcar os estados acessíveis, e a segunda, denominada "considerado", cuja função é indicar que o correspondente estado já foi levado em conta pelo método. O Algoritmo 3.10 apresenta um método prático que sistematiza a identificação de estados inacessíveis.

Algoritmo 3.10 (Eliminação de estados inacessíveis, versão 2) *Obtenção de um autômato sem estados inacessíveis equivalente a outro com estados inacessíveis.*

- Entrada: um autômato finito $M = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$, representado na notação tabular;
- Saída: um autômato finito $N = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$, isento de estados inacessíveis, e tal que $L(N) = L(M)$;
- Método:

  1. Considere-se a linha correspondente ao estado inicial de M. Marque-se este estado como "acessível" na coluna apropriada.
  2. Para cada célula, desta linha da tabela, que contiver o nome de algum estado, marcar tal estado, na linha correspondente, como "acessível". Por último, marque-se o estado corrente como "considerado".
  3. Escolha-se arbitrariamente qualquer outro estado que tenha sido previamente marcado como "acessível", mas que ainda não esteja marcado como "considerado". Repitam-se os passos (2) e (3) até que não mais existam na tabela estados acessíveis, porém não considerados.
  4. N é definido a partir de M eliminando-se todos os estados inacessíveis (ou seja, aqueles que não foram marcados como acessíveis pelo algoritmo), bem como todas as transições que deles partem ou a eles chegam.

Exemplo 3.24 Seja $M = (Q, \Sigma, \delta, q_0, F)$, representado através da Tabela 3.16.

	δ	a	b	c	d	e	f	g
$\rightarrow$	q_0	q_0	q_4	q_3				
	q_1	q_4			q_1			
$\leftarrow$	q_2		q_4			q_1		
	q_3					q_4		
	q_4				q_3		q_5	
$\leftarrow$	q_5			q_0				q_5

Tabela 3.16: Autômato original com estados inacessíveis

Iniciando o procedimento:

- Marca-se q_0 como acessível

	δ	a	b	c	d	e	f	g	Acessível	Considerado
$\rightarrow$	q_0	q_0	q_4	q_3					✓	
	q_1	q_4			q_1					
$\leftarrow$	q_2		q_4			q_1				
	q_3					q_4				
	q_4				q_3		q_5			
$\leftarrow$	q_5			q_0				q_5		

Tabela 3.17: Autômato original com estados inacessíveis, q_0 acessível

Tratando o (único) estado acessível ainda não considerado q_0 (o qual referencia q_0, q_4 e q_3):

- q_0 já estava marcado como "acessível"
- q_4 é marcado como "acessível"
- q_3 também é marcado como "acessível"
- Ao final, q_0 é marcado como "considerado"

	δ	a	b	c	d	e	f	g	Acessível	Considerado
$\rightarrow$	q_0	q_0	q_4	q_3					✓	✓
	q_1	q_4			q_1					
$\leftarrow$	q_2		q_4			q_1				
	q_3					q_4			✓	
	q_4				q_3		q_5		✓	
$\leftarrow$	q_5			q_0				q_5		

Tabela 3.18: Autômato original com estados inacessíveis, q_0 considerado

Tratando (arbitrariamente) o estado acessível não-considerado q_3 (o qual referencia q_4):

- q_4 já estava marcado como "acessível"
- Ao final, q_3 é marcado como "considerado"

	δ	a	b	c	d	e	f	g	Acessível	Considerado
$\rightarrow$	q_0	q_0	q_4	q_3					✓	✓
	q_1	q_4			q_1					
$\leftarrow$	q_2		q_4			q_1				
	q_3					q_4			✓	✓
	q_4				q_3		q_5		✓	
$\leftarrow$	q_5			q_0				q_5		

Tabela 3.19: Autômato original com estados inacessíveis, q_3 considerado

Tratando o (único) estado acessível não-considerado q_4 (o qual referencia q_3 e q_5):

- q_3 já estava marcado como "acessível"
- q_5 é marcado como "acessível"
- Ao final, q_4 é marcado como "considerado"

	δ	a	b	c	d	e	f	g	Acessível	Considerado
$\rightarrow$	q_0	q_0	q_4	q_3					✓	✓
	q_1	q_4			q_1					
$\leftarrow$	q_2		q_4			q_1				
	q_3					q_4			✓	✓
	q_4				q_3		q_5		✓	✓
$\leftarrow$	q_5			q_0				q_5	✓	

Tabela 3.20: Autômato original com estados inacessíveis, q_4 considerado

Tratando o (único) estado acessível não-considerado q_5 (o qual referencia q_0 e q_5):

- q_0 já estava marcado como "acessível"
- q_5 também já estava marcado como "acessível"
- Ao final, q_5 é marcado como "considerado"

	δ	a	b	c	d	e	f	g	Acessível	Considerado
$\rightarrow$	q_0	q_0	q_4	q_3					✓	✓
	q_1	q_4			q_1					
$\leftarrow$	q_2		q_4			q_1				
	q_3					q_4			✓	✓
	q_4				q_3		q_5		✓	✓
$\leftarrow$	q_5			q_0				q_5	✓	✓

Tabela 3.21: Autômato original com estados inacessíveis, q_5 considerado

Nada mais havendo para fazer, o algoritmo se encerra descartando as duas colunas auxiliares e as linhas correspondentes aos estados q_1 e q_2, que permaneceram sem a marca "acessível" até o final. Resulta o autômato da Tabela 3.22.

	δ	a	b	c	d	e	f	g
$\rightarrow$	q_0	q_0	q_4	q_3				
	q_3					q_4		
	q_4				q_3		q_5	
$\leftarrow$	q_5			q_0				q_5

Tabela 3.22: Autômato equivalente ao da Tabela 3.16, eliminados os estados inacessíveis

□

Estados inúteis são estados que, apesar de poderem ser alcançados a partir do estado inicial do autômato, não conduzem a nenhum de seus estados finais. Logo, eles em nada contribuem para a aceitação de sentenças da linguagem definida pelo autômato, podendo portanto ser removidos sem qualquer prejuízo para a linguagem reconhecida.

Estados inúteis podem ser sistematicamente identificados e eliminados usando-se para isso o Algoritmo 3.11. A base de indução usada no algoritmo de eliminação de estados inúteis será o conjunto dos estados finais do autômato, que, por definição, são sempre úteis (todo estado final reconhece pelo menos uma cadeia, a cadeia vazia). Naturalmente, se não houver estados finais no autômato, todos os seus estados podem ser declarados inúteis, e a linguagem por ele definida será vazia.

Seja $M = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$ o autômato original e $N = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$ o autômato que se deseja obter. O Algoritmo 3.11 permite a obtenção sistemática de N a partir de M.

Algoritmo 3.11 (Eliminação de estados inúteis, versão 1) *Obtenção de um autômato sem estados inúteis equivalente a outro com estados inúteis, porém sem estados inacessíveis.*

- Entrada: um autômato finito $M = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$;
- Saída: um autômato finito $N = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$, isento de estados inúteis, e tal que $L(N) = L(M)$;
- Método:

  1. $Q_{20} \leftarrow \{q_{2i} \mid q_{1i} \in F_1\}$;
  2. $i \leftarrow 1$;
  3. $Q_{2i} \leftarrow Q_{2(i-1)} \cup \{q_{2k} \mid q_{1k} \in Q_1 \text{ e } \delta_1(q_{1k}, \sigma) = q, q \in Q_{i-1}, \sigma \in (\Sigma \cup \{\epsilon\})\}$;
  4. Se $Q_{2i} \neq Q_{2(i-1)}$, então $i \leftarrow i + 1$ e desviar para (3); caso contrário, $Q_2 \leftarrow Q_{2i}$;
  5. $F_2 \leftarrow \{q_{2k} \in Q_2 \mid q_{1k} \in F_1\}$;

6. $\delta_2 \leftarrow \{(q_{2i}, \sigma) \rightarrow q_{2j} \mid (q_{1i}, \sigma) \rightarrow q_{1j} \in \delta_1 \text{ e } q_{2i}, q_{2j} \in Q_2\}$.

A aplicação deste algoritmo pode ser sistematizada de forma semelhante à que foi elaborada para o algoritmo de eliminação de estados inacessíveis. Basta representar a função de transição δ na forma tabular e acrescentar duas novas colunas à tabela: a primeira, denominada "útil", e a segunda, denominada "considerado", e executar os passos do Algoritmo 3.12.

Algoritmo 3.12 (Eliminação de estados inúteis, versão 2) *Método prático para obtenção de um autômato sem estados inúteis equivalente a outro com estados inúteis, porém sem estados inacessíveis.*

- Entrada: um autômato finito $M = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$, representado na notação tabular;
- Saída: um autômato finito $N = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$, isento de estados inúteis, e tal que $L(N) = L(M)$;
- Método:

  1. Considerem-se as linhas correspondentes aos estados finais de M. Marquem-se as mesmas como úteis, na coluna apropriada.
  2. Selecione-se um estado qualquer marcado como "útil", porém ainda não marcado como "considerado". Inspecionem-se todos os demais estados do autômato, identificando quais deles permitem transitar para o estado selecionado. Marquem-se todos como úteis. Marque-se finalmente o estado selecionado como "considerado".
  3. Repita-se o passo (2) até que não reste mais nenhum estado marcado como "útil", mas ainda não como "considerado".
  4. O autômato N é definido a partir de M eliminando-se-lhe todos os estados inúteis, bem como todas as transições que deles partem ou a eles chegam.

Exemplo 3.25 Considere-se o autômato da Figura 3.20, em que todos os estados são acessíveis mas nem todos são úteis.

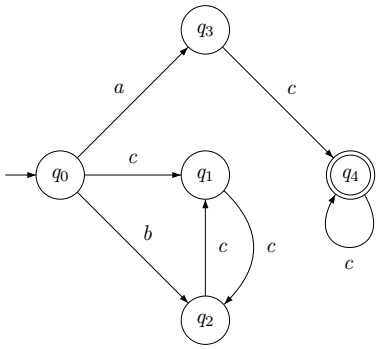

Figura 3.20: Autômato contendo estados inúteis

Conforme o Algoritmo 3.11, obtém-se:

- $Q_0 = \{q_4\}$
- $Q_1 = \{q_3\} \cup \{q_4\}$
- $Q_2 = \{q_0\} \cup \{q_3, q_4\}$
- $Q_3 = Q_2$

Logo, os estados q_1 e q_2 deste autômato são inúteis e podem ser removidos sem prejuízo para a linguagem por ele definida. O autômato resultante torna-se:

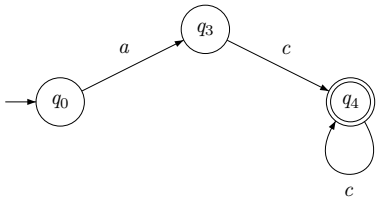

Figura 3.21: Autômato da Figura 3.20 após a remoção dos estados inúteis

Na notação tabular, obtém-se o seguinte resultado, descrito em suas etapas intermediárias. Inicia-se estendendo a tabela de transição com as colunas "útil" e "considerado", e marcando como úteis todos os estados finais (no caso, apenas q_4):

- Marca-se como útil o estado q_4 (final)

	δ	a	b	c	Útil	Considerado
$\rightarrow$	q_0	q_3	q_2	q_1		
	q_1			q_2		
	q_2			q_1		
	q_3			q_4		
$\leftarrow$	q_4			q_4	✓	

Tabela 3.23: Autômato da Figura 3.20, estado q_4 útil

Seleciona-se a linha correspondente ao único estado útil (caso houvesse mais de um, poder-se-ia escolher arbitrariamente dentre eles). Verifica-se por inspeção que q_4 é referenciado por q_4 e q_3. Assim, marca-se q_3 como útil, e q_4 já está marcado. Pode-se então marcar q_4 como considerado.

- As duas únicas referências a q_4 estão nos estados q_3 e q_4;
- Marca-se q_3 como útil;
- q_4 já estava marcado;
- Ao final, marca-se q_4 como considerado.

	δ	a	b	c	Útil	Considerado
$\rightarrow$	q_0	q_3	q_2	q_1		
	q_1			q_2		
	q_2			q_1		
	q_3			q_4	✓	
$\leftarrow$	q_4			q_4	✓	✓

Tabela 3.24: Autômato da Figura 3.20, estado q_4 considerado

Seleciona-se em seguida q_3, o único estado útil não considerado:

- A única referência a q_3 está no estado q_0;
- Marca-se q_0 como útil;
- Ao final, marca-se q_3 como considerado.

	δ	a	b	c	Útil	Considerado
$\rightarrow$	q_0	q_3	q_2	q_1	✓	
	q_1			q_2		
	q_2			q_1		
	q_3			q_4	✓	✓
$\leftarrow$	q_4			q_4	✓	✓

Tabela 3.25: Autômato da Figura 3.20, estado q_0 útil

Seleciona-se q_0, o único estado útil não considerado. Como q_0 não é referenciado por nenhum outro estado, nenhum novo estado útil foi encontrado. Marca-se q_0 como considerado e, não havendo estados úteis não considerados na tabela, o algoritmo se encerra: q_1 e q_2 são estados inúteis:

	δ	a	b	c	Útil	Considerado
$\rightarrow$	q_0	q_3	q_2	q_1	✓	✓
	q_1			q_2		
	q_2			q_1		
	q_3			q_4	✓	✓
$\leftarrow$	q_4			q_4	✓	✓

Tabela 3.26: Autômato da Figura 3.20, estado q_0 considerado

O autômato resultante, sem as colunas auxiliares e sem os estados inúteis, é apresentado na Tabela 3.27. Note-se que as transições $\delta(q_0, b) = q_2$ e $\delta(q_0, c) = q_1$ também foram removidas.

	δ	a	b	c
$\rightarrow$	q_0	q_3		
	q_3			q_4
$\leftarrow$	q_4			q_4

Tabela 3.27: Autômato equivalente ao da Figura 3.20, estados inúteis eliminados

□

Tendo visto os algoritmos de eliminação de transições em vazio, de eliminação de não-determinismos e de eliminação de estados inacessíveis e inúteis, cumpre responder às questões:

1. Dado um autômato finito qualquer, será possível obter uma versão determinística, isenta de transições em vazio e estados inacessíveis ou inúteis, aplicando-se uma só vez cada um dos algoritmos descritos?

2. Em caso afirmativo, em qual seqüência devem eles ser aplicados?

Para responder a essas questões, é suficiente observar que:

- A eliminação de transições em vazio:
  - *Pode introduzir* não-determinismos;
  - *Pode fazer surgir* estados inacessíveis ou inúteis.
- A eliminação de não-determinismos:
  - *Pode fazer surgir* estados inacessíveis ou inúteis;
  - *Não introduz* transições em vazio.
- A eliminação de estados inacessíveis ou inúteis:
  - *Não faz surgir* não-determinismos;

– *Não introduz* transições em vazio.

Logo, é fácil provar que a resposta para as questões inicialmente propostas é "sim", bastando para isso aplicar os algoritmos na seguinte seqüência:

1. Eliminação de transições em vazio, se houver;
2. Eliminação dos não-determinismos restantes, caso haja algum;
3. Eliminação de estados inacessíveis e inúteis, caso existam.

Qualquer outra ordem poderá, dependendo do autômato que estiver sendo manipulado, acarretar a necessidade de se aplicar um mesmo algoritmo mais de uma vez. Por exemplo, a eliminação de não-determinismos seguida da eliminação de transições em vazio pode fazer com que o autômato se torne novamente não-determinístico, exigindo portanto uma nova aplicação do algoritmo usado anteriormente.

De acordo com as definições previamente apresentadas, é fácil perceber que os autômatos finitos determinísticos isentos de transições em vazio constituem casos particulares do modelo mais geral, os autômatos finitos não-determinísticos com transições em vazio. No entanto, apesar de se tratar de um modelo conceitualmente mais simples, pode-se demonstrar que a todo autômato finito não-determinístico que apresente transições em vazio corresponde um outro autômato finito determinístico que aceita a mesma linguagem.

Desse modo, não-determinismos e transições em vazio não contribuem em nada para aumentar o poder dos autômatos finitos quanto a uma eventual ampliação da classe de linguagens por eles reconhecida. Por esse motivo, nos demais capítulos, serão referenciados apenas os autômatos finitos, de uma forma geral, sem preocupações com a existência de eventuais não-determinismos, transições em vazio, estados inacessíveis ou estados inúteis.

Exemplo 3.26 Considere-se um autômato finito M, conforme a Tabela 3.28, que apresenta transições em vazio e não-determinismo (conseqüência, no caso, das transições em vazio existentes).

	δ	a	b	c	d	e	ϵ
$\rightarrow$	q_0		q_1		q_1		q_2
	q_1	q_3			q_1		q_4
	q_2		q_2		q_3		
	q_3	q_4		q_3			
$\leftarrow$	q_4		q_2			q_4	q_0

Tabela 3.28: Autômato original com transições em vazio

A obtenção de um autômato equivalente, porém sem transições em vazio e determinístico, é feita de acordo com os seguintes passos:

1. *Eliminação de transições em vazio*

   Seleciona-se arbitrariamente uma linha da tabela que descreva o comportamento do autômato em um estado do qual parta alguma transição em vazio. Há três estados nessas condições: q_0, q_1 e q_4, dos quais partem transições em vazio para q_2, q_4 e q_0, respectivamente.

   Escolhendo arbitrariamente q_0 para remover a transição em vazio correspondente, verifica-se que essa transição referencia o estado q_2. Assim, copiam-se todas as transições referenciadas

por q_2 nas células da tabela correspondentes, relativas ao estado q_0, e elimina-se a transição em vazio $(q_0, \epsilon) \rightarrow q_2$.

- Seleciona-se a linha q_0;
- Copiam-se transições para q_2 e q_3 no estado q_0, obtendo-se os conjuntos de estados-destino $\{q_1, q_2\}$ e $\{q_1, q_3\}$, respectivamente, no estado q_0;
- Ao final, elimina-se a transição em vazio $(q_0, \epsilon) \rightarrow q_2$.

O resultado é apresentado na Tabela 3.29.

	δ	a	b	c	d	e	ϵ
$\rightarrow$	q_0		$\{q_1, q_2\}$		$\{q_1, q_3\}$		
	q_1	q_3			q_1		q_4
	q_2		q_2		q_3		
	q_3	q_4		q_3			
$\leftarrow$	q_4		q_2			q_4	q_0

Tabela 3.29: Eliminação de $(q_0, \epsilon) \rightarrow q_2$

Seleciona-se arbitrariamente q_1 dentre q_1 e q_4, que fazem referência em vazio aos estados q_4 e q_0, respectivamente. Observa-se que q_1 faz referência a q_4, e portanto os estados referenciados por q_4 serão copiados nas células correspondentes de q_1, conforme a Tabela 3.30.

	δ	a	b	c	d	e	ϵ
$\rightarrow$	q_0		$\{q_1, q_2\}$		$\{q_1, q_3\}$		
$\leftarrow$	q_1	q_3	q_2		q_1	q_4	q_0
	q_2		q_2		q_3		
	q_3	q_4		q_3			
$\leftarrow$	q_4		q_2			q_4	q_0

Tabela 3.30: Eliminação de $(q_1, \epsilon) \rightarrow q_4$

Note-se que q_1 recebe também de q_4 o atributo de estado final. Na tabela resultante, apesar de ser eliminada a transição em vazio $(q_1, \epsilon) \rightarrow q_4$, o estado q_1 recebeu uma nova transição em vazio $(q_1, \epsilon) \rightarrow q_0$ e, por essa razão, q_1 deve incorporar agora também os estados referenciados por q_0, como mostra a Tabela 3.31.

	δ	a	b	c	d	e	ϵ
$\rightarrow$	q_0		$\{q_1, q_2\}$		$\{q_1, q_3\}$		
$\leftarrow$	q_1	q_3	$\{q_1, q_2\}$		$\{q_1, q_3\}$	q_4	
	q_2		q_2		q_3		
	q_3	q_4		q_3			
$\leftarrow$	q_4		q_2			q_4	q_0

Tabela 3.31: Eliminação de $(q_1, \epsilon) \rightarrow q_0$

Resta a transição em vazio $(q_4, \epsilon) \rightarrow q_0$, que é eliminada analogamente, como mostra a Tabela 3.32.

	δ	a	b	c	d	e	ϵ
$\rightarrow$	q_0		$\{q_1, q_2\}$		$\{q_1, q_3\}$		
$\leftarrow$	q_1	q_3	$\{q_1, q_2\}$		$\{q_1, q_3\}$	q_4	
	q_2		q_2		q_3		
	q_3	q_4		q_3			
$\leftarrow$	q_4		$\{q_1, q_2\}$		$\{q_1, q_3\}$	q_4	

Tabela 3.32: Eliminação de $(q_4, \epsilon) \rightarrow q_0$

Resulta finalmente o autômato não-determinístico da Tabela 3.33.

	δ	a	b	c	d	e
$\rightarrow$	q_0		$\{q_1, q_2\}$		$\{q_1, q_3\}$	
$\leftarrow$	q_1	q_3	$\{q_1, q_2\}$		$\{q_1, q_3\}$	q_4
	q_2		q_2		q_3	
	q_3	q_4		q_3		
$\leftarrow$	q_4		$\{q_1, q_2\}$		$\{q_1, q_3\}$	q_4

Tabela 3.33: Autômato equivalente ao da Tabela 3.28, eliminadas as transições em vazio

2. *Eliminação de não-determinismos*

   Através da inspeção da função de transição, torna-se evidente a necessidade de criação imediata de dois novos estados: um representando o conjunto $\{q_1, q_2\}$ e outro representando o conjunto $\{q_1, q_3\}$. Após a incorporação desses novos estados ao autômato, bem como das transições que partem de cada um dos estados individualmente considerados, obtém-se o autômato da Tabela 3.34.

	δ	a	b	c	d	e
$\rightarrow$	q_0		$\{q_1, q_2\}$		$\{q_1, q_3\}$	
$\leftarrow$	q_1	q_3	$\{q_1, q_2\}$		$\{q_1, q_3\}$	q_4
	q_2		q_2		q_3	
	q_3	q_4		q_3		
$\leftarrow$	q_4		$\{q_1, q_2\}$		$\{q_1, q_3\}$	q_4
$\leftarrow$	$\{q_1, q_2\}$	q_3	$\{q_1, q_2\}$		$\{q_1, q_3\}$	q_4
$\leftarrow$	$\{q_1, q_3\}$	$\{q_3, q_4\}$	$\{q_1, q_2\}$	q_3	$\{q_1, q_3\}$	q_4

Tabela 3.34: Autômato isento de não-determinismos, versão inicial

Ao efetuar a combinação das transições que emergem de q_1 e q_3, deve-se observar o surgimento de um novo não-determinismo anteriormente inexistente no autômato. O mesmo corresponde a uma transição não-determinística para q_3 ou q_4, quando a entrada é a, e pode ser eliminada através da criação de um novo estado $\{q_3, q_4\}$, conforme a Tabela 3.35.

	δ	a	b	c	d	e
$\rightarrow$	q_0		$q_1\,q_2$		$q_1\,q_3$	
$\leftarrow$	q_1	q_3	$q_1\,q_2$		$q_1\,q_3$	q_4
	q_2		q_2		q_3	
	q_3	q_4		q_3		
$\leftarrow$	q_4		$q_1\,q_2$		$q_1\,q_3$	q_4
$\leftarrow$	$q_1\,q_2$	q_3	$q_1\,q_2$		$q_1\,q_3$	q_4
$\leftarrow$	$q_1\,q_3$	$q_3\,q_4$	$q_1\,q_2$	q_3	$q_1\,q_3$	q_4
$\leftarrow$	$q_3\,q_4$	q_4	$q_1\,q_2$	q_3	$q_1\,q_3$	q_4

Tabela 3.35: Autômato isento de não-determinismos, versão final

3. *Estados renomeados*

   Renomeando-se os estados, a fim de aumentar a legibilidade da Tabela 3.35, obtém-se a Tabela 3.36.

	δ	a	b	c	d	e
$\rightarrow$	q_0		q_5		q_6	
$\leftarrow$	q_1	q_3	q_5		q_6	q_4
	q_2		q_2		q_3	
	q_3	q_4		q_3		
$\leftarrow$	q_4		q_5		q_6	q_4
$\leftarrow$	q_5	q_3	q_5		q_6	q_4
$\leftarrow$	q_6	q_7	q_5	q_3	q_6	q_4
$\leftarrow$	q_7	q_4	q_5	q_3	q_6	q_4

Tabela 3.36: Autômato da Tabela 3.35, com estados renomeados

4. *Eliminação de estados inacessíveis*

   Através da marcação dos estados acessíveis e considerados, obtém-se a Tabela 3.37.

	δ	a	b	c	d	e	Acessível	Considerado
$\rightarrow$	q_0		q_5		q_6		✓	✓
$\leftarrow$	q_1	q_3	q_5		q_6	q_4		
	q_2		q_2		q_3			
	q_3	q_4		q_3			✓	✓
$\leftarrow$	q_4		q_5		q_6	q_4	✓	✓
$\leftarrow$	q_5	q_3	q_5		q_6	q_4	✓	✓
$\leftarrow$	q_6	q_7	q_5	q_3	q_6	q_4	✓	✓
$\leftarrow$	q_7	q_4	q_5	q_3	q_6	q_4	✓	✓

Tabela 3.37: Autômato da Tabela 3.36, logo após a execução do algoritmo de eliminação de estados inacessíveis

ou, simplesmente:

	δ	a	b	c	d	e
$\rightarrow$	q_0		q_5		q_6	
	q_3	q_4		q_3		
$\leftarrow$	q_4		q_5		q_6	q_4
$\leftarrow$	q_5	q_3	q_5		q_6	q_4
$\leftarrow$	q_6	q_7	q_5	q_3	q_6	q_4
$\leftarrow$	q_7	q_4	q_5	q_3	q_6	q_4

Tabela 3.38: Autômato final obtido pela eliminação de transições em vazio, não-determinismos e estados inacessíveis

Observe-se que o autômato resultante não possui estados inúteis, sendo portanto desnecessária a aplicação do algoritmo de eliminação de estados inúteis. □

Um importante aspecto de extensão que costuma ser considerado no estudo dos autômatos finitos diz respeito à possibilidade de movimentação do cursor de leitura em ambos os sentidos, em contraste com a movimentação em um só sentido, mais freqüentemente adotada na literatura sobre o assunto.

Os autômatos finitos, por definição, efetuam a leitura de símbolos na fita de entrada em apenas um sentido (normalmente da esquerda para a direita). A questão que surge é: se fosse permitido aos autômatos finitos deslocar o cursor de leitura em ambos os sentidos, será que isso os tornaria mais potentes, aptos a reconhecer uma classe de linguagens mais complexa que a das linguagens regulares? A resposta é negativa: é possível demonstrar (ver [46]) que a classe de linguagens reconhecida pelos autômatos finitos de "duplo sentido" é exatamente a mesma que é aceita pelos de sentido único, ou seja, as linguagens do tipo 3.

Para finalizar este item, algumas considerações sobre a importância prática dos autômatos finitos. Conforme mencionado anteriormente, as linguagens regulares são muito utilizadas na especificação de linguagens de programação e como linguagens para a especificação de dados em ambientes interativos, tais como editores de texto e linguagens de comando. Os autômatos finitos representam, nesse caso, os modelos de implementação para tais linguagens, sendo largamente empregados em compiladores e outras ferramentas de apoio ao desenvolvimento de programas e ao processamento de dados.

A profusão e a importância dos resultados teóricos existentes, referentes a esta classe de reconhecedores, torna-os especialmente atraentes em aplicações práticas, uma vez que eles permitem a obtenção de programas muito eficientes e compactos, e ao mesmo tempo adequadamente estruturados e autodocumentados. Devido a essas características interessantes, diversas ferramentas têm sido desenvolvidas para a geração automática de programas e/ou de circuitos seqüenciais que realizam abstrações representadas pelos autômatos finitos, obtidos a partir de notações tais como as expressões regulares. Essas ferramentas vêm sendo cada vez mais utilizadas por profissionais de computação, tornando seu trabalho mais abrangente, simples, rápido e confiável.

3.4 Equivalência entre Gramáticas Regulares e Conjuntos Regulares

As linguagens regulares foram definidas no Capítulo 2 como sendo aquelas geradas por gramáticas lineares, em particular as gramáticas lineares à direita. Inicialmente foi apresentada a definição dos conjuntos regulares, tendo sido também definidas as expressões regulares, que constituem uma notação mais cômoda para a representação de tais conjuntos. Em seguida foram caracterizados os autômatos finitos como mecanismos de reconhecimento para os conjuntos regulares.

O objetivo das Seções 3.4 e 3.5 é discutir e demonstrar a plena equivalência, no que diz respeito à classe de linguagens que são capazes de representar, dos diversos tipos de formalismos até aqui estudados: os conjuntos (expressões) regulares, as gramáticas lineares à direita e os autômatos finitos.

Na presente seção é mostrada a equivalência entre as linguagens caracterizadas pelos conjuntos regulares e as geradas pelas gramáticas lineares à direita, ou seja, prova-se que toda e qualquer linguagem gerada por alguma gramática linear é um conjunto regular, e

também, inversamente, que todo e qualquer conjunto regular pode ser expresso através de uma gramática linear.

Para efeitos práticos, estudam-se aqui apenas as gramáticas lineares à direita, porém os mesmos resultados se aplicam e podem ser deduzidos igualmente para as gramáticas lineares à esquerda. O termo **linguagem linear à direita (esquerda)** será empregado para denotar linguagens geradas por gramáticas lineares à direita (esquerda).

A Figura 3.22 destaca as equivalências estudadas nesta seção.

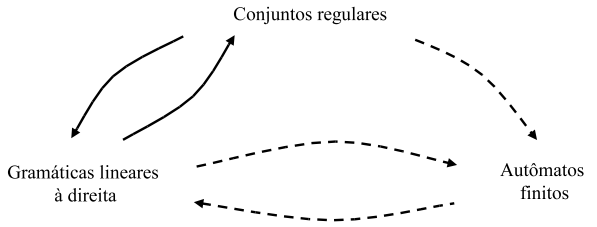

Figura 3.22: Equivalência dos formalismos — parte 1

Teorema 3.6 (Conjuntos regulares ⇒ gramáticas lineares à direita) *Todo conjunto regular é gerado por uma gramática linear à direita.*

Justificativa Deseja-se demonstrar que todo e qualquer conjunto regular define uma linguagem que também pode ser gerada por uma gramática linear à direita.

Por definição, $\varnothing, \{\epsilon\}, \{\sigma\}, \sigma \in \Sigma$, onde Σ é um alfabeto (conjunto finito e não-vazio), são conjuntos regulares. Da mesma forma, $X \cup Y, XY$ e X^*, com X e Y conjuntos regulares, também são conjuntos regulares. A equivalência de tais conjuntos regulares com as correspondentes gramáticas lineares à direita que os geram é mostrada a seguir:

- $\varnothing$ é uma linguagem linear à direita, pois:

  $G = (\{S\} \cup \Sigma, \Sigma, \varnothing, S)$ é tal que $L(G) = \varnothing$

- $\{\epsilon\}$ é uma linguagem linear à direita, pois:

  $G = (\{S\} \cup \Sigma, \Sigma, \{S \to \epsilon\}, S)$ é tal que $L(G) = \{\epsilon\}$

- $\{\sigma\}$ é uma linguagem linear à direita, pois:

  $G = (\{\sigma, S\}, \{\sigma\}, \{S \to \sigma\}, S)$ é tal que $L(G) = \{\sigma\}$

  Suponha-se agora que X e Y sejam dois conjuntos regulares, gerados por gramáticas lineares à direita. É válido, nesse caso, admitir que X e Y sejam gerados pelas gramáticas G_x e G_y:

$$\begin{aligned} X &= L(G_x), G_x = (\Sigma_x \cup N_x, \Sigma_x, P_x, S_x) \\ Y &= L(G_y), G_y = (\Sigma_y \cup N_y, \Sigma_y, P_y, S_y) \end{aligned}$$

  Admita-se ainda, sem perda de generalidade, que $N_x \cap N_y = \varnothing$. Caso isso não seja verdadeiro, podem-se renomear os não-terminais, de modo que essa condição seja satisfeita. Nessa situação, a aplicação das operações de união, concatenação

e fechamento reflexivo sobre X e Y geram novas linguagens lineares à direita, as quais podem ser representadas, com base em G_x e G_y, da seguinte forma:

- $Z = X \cup Y$ é uma linguagem linear à direita, pois:

  $G_z = (\Sigma_x \cup \Sigma_y \cup N_x \cup N_y \cup \{S_z\}, \Sigma_x \cup \Sigma_y, P_x \cup P_y \cup \{S_z \rightarrow S_x, S_z \rightarrow S_y\}, S_z)$ é tal que $L(G_z) = X \cup Y$.

- $Z = XY$ é uma linguagem linear à direita, pois:

  $G_z = (\Sigma_x \cup \Sigma_y \cup N_x \cup N_y, \Sigma_x \cup \Sigma_y, P_y \cup P_z, S_x)$, sendo P_z obtido pela aplicação das regras:

  a) Se $A \rightarrow \sigma B \in P_x$, então $A \rightarrow \sigma B \in P_z$

  b) Se $A \rightarrow \sigma \in P_x$, então $A \rightarrow \sigma S_y \in P_z$

  é tal que $L(G_z) = XY$.

- $Z = X^*$ é uma linguagem linear à direita, pois:

  $G_z = (\Sigma_x \cup N_x \cup \{S_z\}, \Sigma_x, \{S_z \rightarrow S_x, S_z \rightarrow \epsilon\} \cup P_z, S_z)$, sendo P_z obtido pela aplicação das regras:

  a) Se $A \rightarrow \sigma B \in P_x$, então $A \rightarrow \sigma B \in P_z$

  b) Se $A \rightarrow \sigma \in P_x$, então $A \rightarrow \sigma S_z \in P_z$

  é tal que $L(G_z) = X^*$.

■

As regras acima relacionadas podem ser entendidas da seguinte forma: a união de duas linguagens lineares à direita é representada através de uma gramática em que uma nova raiz e duas novas produções são introduzidas, sendo que cada alternativa de substituição para a nova raiz conduz à geração das sentenças de um ou do outro conjunto original; a concatenação é obtida fazendo-se com que, ao término da derivação de sentenças pertencentes ao primeiro conjunto, seja ativado o não-terminal que representa a raiz da gramática do segundo conjunto, possibilitando a justaposição da cadeia já formada com aquela que será gerada pela segunda gramática; finalmente, o fechamento reflexivo e transitivo é representado, em termos gramaticais, de maneira similar ao caso anterior: ao término de uma derivação, reativa-se o não-terminal que representa a raiz da gramática, de tal forma que seja permitida a concatenação de novas sentenças geradas pela mesma gramática (note-se a necessidade da produção que gera a cadeia vazia a partir da raiz).

Exemplo 3.27 Considere-se o alfabeto $\Sigma = \{0, 1, 2\}$. Os conjuntos regulares $L_0 = \{0\}, L_1 = \{1\}, L_2 = \{2\}$ são definidos, respectivamente, pelas gramáticas:

- $G_0 = (\{0, S_0\}, \{0\}, \{S_0 \rightarrow 0\}, S_0)$
- $G_1 = (\{1, S_1\}, \{1\}, \{S_1 \rightarrow 1\}, S_1)$
- $G_2 = (\{2, S_2\}, \{2\}, \{S_2 \rightarrow 2\}, S_2)$

Através da aplicação das regras correspondentes ao fechamento reflexivo e transitivo, obtêm-se, a partir de G_0 e G_1, novas gramáticas G_3 e G_4 representando, respectivamente, os conjuntos $\{0\}^*$ e $\{1\}^*$:

- $G_3 = (\{0, S_0, S_3\}, \{0\}, \{S_3 \to S_0, S_3 \to \epsilon, S_0 \to 0S_3\}, S_3)$
- $G_4 = (\{1, S_1, S_4\}, \{1\}, \{S_4 \to S_1, S_4 \to \epsilon, S_1 \to 1S_4\}, S_4)$

Da concatenação dos conjuntos $\{0\}^*$ e $\{1\}^*$ resulta o conjunto regular $\{0\}^*\{1\}^*$, representado através da gramática linear à direita G_5:

- $G_5 = (\{0, 1, S_0, S_1, S_3, S_4\}, \{0, 1\}, P_5, S_3)$
  $P_5 = \{\underbrace{S_3 \to S_0, S_3 \to S_4, S_0 \to 0S_3}_{G_3}, \underbrace{S_4 \to S_1, S_4 \to \epsilon, S_1 \to 1S_4}_{G_4}\}$

Finalmente, a linguagem obtida pela união de $L(G_5)$ com $L(G_2)$ pode ser representada pela gramática linear G_6:

- $G_6 = (\{0, 1, 2, S_0, S_1, S_2, S_3, S_4, S_6\}, \{0, 1, 2\}, P_6, S_6)$
  $P_6 = \{S_6 \to S_2, S_6 \to S_3, \underbrace{S_2 \to 2}_{G_2},$
  $\underbrace{S_3 \to S_0, S_3 \to S_4, S_0 \to 0S_3, S_4 \to S_1, S_4 \to \epsilon, S_1 \to 1S_4}_{G_5}\}$

  Portanto, $L(G_6) = \{0\}^*\{1\}^* \cup \{2\}$. □

Por se tratar de um método canônico de manipulação, a gramática resultante, como conseqüência da aplicação da técnica exposta, nem sempre corresponderá, necessariamente, a uma versão muito concisa ou intuitiva, devendo-se efetuar em seguida as simplificações julgadas necessárias.

Exemplo 3.28 A gramática G_6 do Exemplo 3.27 poderia ser simplificada, mantendo-se a linguagem original e reduzindo-se a quantidade de símbolos não-terminais e o número de produções. Uma possibilidade seria:

- $G_6 = (\{0, 1, 2, S, A, B\}, \{0, 1, 2\}, P_6, S)$
  $P_6 = \{\underbrace{S \to 2}_{\{2\}}, \underbrace{S \to A, A \to 0A}_{\{0\}^*}, \underbrace{A \to B}_{\{0\}^*\{1\}^*}, \underbrace{B \to 1B, B \to \epsilon}_{\{1\}^*}\}$

□

Teorema 3.7 (Conjuntos regulares ⇐ gramáticas lineares à direita) *Toda gramática linear à direita gera um conjunto regular.*

Justificativa Mostra-se agora a proposição inversa, ou seja, que toda e qualquer linguagem gerada por uma gramática linear à direita constitui um conjunto regular. Para tanto, deve-se lembrar que gramáticas lineares à direita se caracterizam por apresentarem apenas produções com os formatos seguintes:

- $X_i \to \sigma X_j$
- $X_i \to X_j$
- $X_i \to \sigma$
- $X_i \to \epsilon$

com $\sigma \in \Sigma$ e $X_i, X_j \in N$.

O Algoritmo 3.13, a seguir apresentado, permite a obtenção, de forma sistemática, de uma expressão que representa o conjunto regular definido por uma gramática linear à direita fornecida como entrada.

Inicialmente, a representação algébrica da gramática deve ser modificada, para permitir a representação explícita dos conjuntos denotados pelos seus símbolos não-terminais:

- $X_i \rightarrow \sigma X_j$ torna-se $X_i = \{\sigma\}X_j$
- $X_i \rightarrow X_j$ torna-se $X_i = \{\epsilon\}X_j$
- $X_i \rightarrow \sigma$ torna-se $X_i = \{\sigma\}$
- $X_i \rightarrow \epsilon$ torna-se $X_i = \{\epsilon\}$

Note-se, no primeiro caso, que a expressão $\{\sigma\}X_j$ denota a concatenação da cadeia elementar σ com os elementos do conjunto X_j, disso resultando o conjunto X_i. Tal interpretação é compatível, portanto, com o significado atribuído à produção algébrica correspondente. Considerações semelhantes valem nos demais casos. Uma vez efetuada essa conversão de notação, devem-se agrupar todas as alternativas de substituição para um mesmo símbolo não-terminal X_i, utilizando para isso o operador de união.

Como resultado, obtém-se um **sistema de equações regulares**, em que cada equação corresponde a uma diferente variável (não-terminal) da gramática original. A solução desse sistema de m variáveis e m equações fornece as expressões que definem os conjuntos regulares representados pelos diversos símbolos não-terminais. Em particular, o conjunto regular associado ao não-terminal raiz da gramática define a linguagem por ela descrita. Nesse sistema, cada equação possui o seguinte formato geral:

$$X_i = A_{i1}X_1 \cup A_{i2}X_2 \cup ... \cup A_{im}X_m \cup B_{i1} \cup B_{i2} \cup ...B_{ik}$$

onde:

- $A_{ij} = \{\sigma_{ij}\}$ se $X_i \rightarrow \sigma_{ij}X_j \in P, \sigma_{ij} \in (\Sigma \cup \{\epsilon\})$, ou $A_{ij} = \varnothing$ em caso contrário;
- $B_{ij} = \{\sigma_{ij}\}$ se $X_i \rightarrow \sigma_{ij} \in P, \sigma_{ij} \in (\Sigma \cup \{\epsilon\})$.

O sistema de equações propriamente dito assume então a forma genérica:

$$\begin{array}{rcl} X_1 & = & A_{11}X_1 \cup A_{12}X_2 \cup ... \cup A_{1m}X_m \cup B_{11} \cup B_{12} \cup ...B_{1p} \\ X_2 & = & A_{21}X_1 \cup A_{22}X_2 \cup ... \cup A_{2m}X_m \cup B_{21} \cup B_{22} \cup ...B_{2q} \\ ... & & ... \\ X_m & = & A_{m1}X_1 \cup A_{m2}X_2 \cup ... \cup A_{mm}X_m \cup B_{m1} \cup B_{m2} \cup ...B_{mr} \end{array}$$

Observe-se que cada equação assim obtida pode conter referências a todos os símbolos não-terminais da gramática (X_1 a X_m) bem como a termos constantes (B_{ij}), em quantidade variável conforme a equação.

A seguir, coloca-se em evidência o símbolo não-terminal X_i definido em cada equação. Isso, e o agrupamento de todos os termos que não dependem de X_i, pela aplicação

da propriedade associativa da união dos conjuntos, altera a representação da equação para:

$$X_i = A_{ii}X_i \cup (A_i \cup B_i)$$

onde:

- $A_i = \cup A_{ij}X_j, 1 \leq j \leq m, j \neq i$
- $B_i = B_{i1} \cup B_{i2} \cup ...B_{ik}$

Fazendo-se $C_i = A_i \cup B_i$, são obtidas expressões do tipo:

$$X_i = A_{ii}X_i \cup C_i$$

onde C_i representa todos os termos da i-ésima equação que não contêm referências diretas ao símbolo não-terminal X_i, e A_{ii} representa o conjunto de todas as cadeias sobre Σ que prefixam X_i na equação em que este não-terminal é definido.

É possível demonstrar, por indução, que equações com o formato genérico $X_i = A_{ii}X_i \cup C_i$ possuem, como solução geral, o conjunto:

$$X_i = A_{ii}^* C_i$$

Nesta expressão, não existem ocorrências do símbolo X_i. De fato, substituindo-se X_i por $A_{ii}^* C_i$ na segunda ocorrência deste símbolo em $X_i = A_{ii}X_i \cup C_i$, obtém-se a identidade:

$$X_i = A_{ii}X_i \cup C_i = A_{ii}(A_{ii}^* C_i) \cup C_i = A_{ii}^+ C_i \cup C_i = A_{ii}^* C_i$$

Teorema 3.8 (Solução de $X = AX \cup C$) *A equação regular $X = AX \cup C$ tem como solução geral o conjunto regular $X = A^*C$.*

Justificativa Partindo-se de $X = AX \cup C$, pode-se levantar o conjunto de valores da variável X que satisfazem a essa igualdade. Há dois caminhos possíveis:

1. $X = C$, que é uma solução é trivial, obtida pela simples inspeção de $X = AX \cup C$;
2. $X = AX$, que deve ser assim interpretada: "para obter um novo valor de X, utiliza-se algum valor já conhecido, e concatene-se-lhe um A à esquerda". Aplicando-se essa interpretação à solução obtida em (1), tem-se: $X = AC$, que passa a ser uma nova solução conhecida. Aplicando-se outra vez essa interpretação à solução $X = AC$, obtém-se $X = AAC$, e assim por diante. Note-se que dessa forma foi obtida uma seqüência de soluções progressivamente mais longas, partindo-se da solução trivial, através da concatenação de elementos A à esquerda. Generalizando-se, tem-se a forma $X = A^*C$ como solução geral da equação $X = AX \cup C$.

■

Uma vez obtido o sistema de equações regulares no formato $X_i = A_{ii}^* C_i$, a solução do mesmo é dada pelo Algoritmo 3.13.

Algoritmo 3.13 (Equações regulares) *Resolução de um sistema de equações regulares.*

- Entrada: Uma série de equações regulares sobre variáveis $X_i, 1 \leq i \leq m$, cujos coeficientes A_{ij} são conjuntos regulares sobre um alfabeto Σ.

- Saída: Uma série de conjuntos regulares γ_i sobre Σ, de tal forma que $X_i = \gamma_i$.
- Método:

  1. $i \leftarrow 1$;
  2. Se $i = m$, então desviar para (4). Caso contrário, considerar a i-ésima equação na forma $X_i = A_{ii}X_i \cup C_i$ e substituir todas as ocorrências de X_i por $A_{ii}^* C_i$ nas equações referentes às variáveis $X_j, (i+1) \leq j \leq m$;
  3. $i \leftarrow i + 1$ e desviar para (2);
  4. Se $i = 0$, então FIM. Caso contrário, $X_i \leftarrow \gamma_i = A_{ii}^* C_i$ e substituir todas as ocorrências de X_i pelo conjunto γ_i nas equações referentes às variáveis $X_j, 1 \leq j \leq (i-1)$.
  5. $i \leftarrow i - 1$ e desviar para (4).

■

Este algoritmo opera em dois passos. No primeiro, em que as equações são percorridas em ordem crescente de índices, eliminam-se, de cada equação relativa a uma variável X_i distinta, todas as referências às demais variáveis $X_j, j < i$. Assim, obtém-se um novo conjunto de equações em que, para cada variável X_i do lado esquerdo da equação, existem referências apenas a variáveis $X_j, j \geq i$, ou então a elementos de Σ, do lado direito. Note-se, em particular, que a m-ésima equação contém apenas referências a elementos de Σ à própria variável X_m.

No passo seguinte do algoritmo, efetuado na ordem decrescente dos índices das equações, são eliminadas todas as referências às variáveis do sistema, as quais são substituídas, a partir da m-ésima equação, por conjuntos regulares definidos exclusivamente sobre Σ. Ao término da execução do algoritmo, são obtidos, dessa maneira, conjuntos regulares sobre Σ, que definem a linguagem gerada pelas diversas variáveis do sistema de equações (correspondentes aos símbolos não-terminais da gramática original). Devido ao fato de terem sido empregadas apenas operações que preservam a regularidade dos conjuntos manipulados, é fácil demonstrar que os conjuntos γ_i são regulares, o que mostra, intuitivamente, que toda e qualquer gramática linear à direita gera uma linguagem regular.

Exemplo 3.29 Considere-se a gramática linear à direita G_0:

$$\begin{aligned} G_0 &= (\{a, b, c, X_0, X_1, X_2\}, \{a, b, c, d\}, P_0, X_0) \\ P_0 &= \{X_0 \rightarrow aX_0, X_0 \rightarrow aX_1, X_0 \rightarrow b, \\ &\quad X_1 \rightarrow bX_1, X_1 \rightarrow cX_1, X_1 \rightarrow cX_2, X_1 \rightarrow d, \\ &\quad X_2 \rightarrow aX_0, X_2 \rightarrow bX_1, X_2 \rightarrow cX_2, X_2 \rightarrow c, X_2 \rightarrow d\} \end{aligned}$$

Convertendo-se as produções algébricas para a notação de conjuntos, obtém-se:

- $X_0 = \{a\}X_0, X_0 = \{a\}X_1, X_0 = \{b\}$,
- $X_1 = \{b\}X_1, X_1 = \{c\}X_1, X_1 = \{c\}X_2, X_1 = \{d\}$,
- $X_2 = \{a\}X_0, X_2 = \{b\}X_1, X_2 = \{c\}X_2, X_2 = \{c\}, X_2 = \{d\}$

Do agrupamento das alternativas de substituição de cada símbolo não-terminal resulta o sistema de equações seguinte:

- $X_0 = \{a\}X_0 \cup \{a\}X_1 \cup \{b\}$
- $X_1 = \{b\}X_1 \cup \{c\}X_1 \cup \{c\}X_2 \cup \{d\}$
- $X_2 = \{a\}X_0 \cup \{b\}X_1 \cup \{c\}X_2 \cup \{c\} \cup \{d\}$

Colocando-se em evidência os símbolos não-terminais definidos em cada equação, o novo sistema toma o seguinte aspecto:

- $X_0 = \{a\}X_0 \cup (\{a\}X_1 \cup \{b\}) = A_{00}X_0 \cup C_0$
- $X_1 = \{b, c\}X_1 \cup (\{c\}X_2 \cup \{d\}) = A_{11}X_1 \cup C_1$
- $X_2 = \{c\}X_2 \cup (\{a\}X_0 \cup \{b\}X_1 \cup \{c, d\}) = A_{22}X_2 \cup C_2$

com:

- $A_{00} = \{a\}, A_0 = \{a\}X_1, B_0 = \{b\}$
- $A_{11} = \{b, c\}, A_1 = \{c\}X_2, B_1 = \{d\}$
- $A_{22} = \{c\}, A_2 = \{a\}X_0 \cup \{b\}X_1, B_2 = \{c, d\}$

e:

- $C_0 = A_0 \cup B_0 = \{a\}X_1 \cup \{b\}$
- $C_1 = A_1 \cup B_1 = \{c\}X_2 \cup \{d\}$
- $C_2 = A_2 \cup B_2 = (\{a\}X_0 \cup \{b\}X_1) \cup \{c, d\}$

Eliminando-se as referências aos símbolos não-terminais $X_i, 0 \leq i \leq 2$, nas equações em que os mesmos são definidos, resulta:

- $X_0 = \{a\}^*(\{a\}X_1 \cup \{b\})$
- $X_1 = \{b, c\}^*(\{c\}X_2 \cup \{d\})$
- $X_2 = \{c\}^*(\{a\}X_0 \cup \{b\}X_1 \cup \{c, d\})$

A aplicação do algoritmo de cálculo dos conjuntos regulares definidos pelos símbolos não-terminais da gramática é feita conforme apresentado a seguir.

- Passos 1 e 2 do algoritmo, primeira passagem:

$$
\begin{aligned}
X_0 &= \{a\}^*(\{a\}X_1 \cup \{b\}) \text{ e substituição nas equações de } X_1 \text{ e } X_2: \\
X_0 &= \{a\}^*(\{a\}X_1 \cup \{b\}) \\
X_1 &= \{b, c\}X_1 \cup (\{c\}X_2 \cup \{d\}) \\
X_2 &= \{c\}X_2 \cup (\{a\}(\{a\}^*(\{a\}X_1 \cup \{b\})) \cup \{b\}X_1 \cup \{c, d\}) \\
&= \{c\}X_2 \cup (\{a\}(\{a\}^*\{a\}X_1 \cup \{a\}^*\{b\}) \cup \{b\}X_1 \cup \{c, d\}) \\
&= \{c\}X_2 \cup (\{a\}\{a\}^*\{a\}X_1 \cup \{a\}\{a\} * \{b\} \cup \{b\}X_1 \cup \{c, d\}) \\
&= \{c\}X_2 \cup \{a\}\{a\}^*\{a\}X_1 \cup \{a\}\{a\}^*\{b\} \cup \{b\}X_1 \cup \{c, d\} \\
&= \{c\}X_2 \cup \{a\}\{a\}^+X_1 \cup \{a\}^+\{b\} \cup \{b\}X_1 \cup \{c, d\} \\
&= \{c\}X_2 \cup (\{a\}\{a\}^+ \cup \{b\})X_1 \cup (\{a\}^+\{b\} \cup \{c, d\})
\end{aligned}
$$

- Passos 1 e 2 do algoritmo, segunda passagem:

$$
\begin{aligned}
X_1 &= \{b,c\}^*(\{c\}X_2 \cup \{d\}) \text{ e substituição na equação de } X_2: \\
X_0 &= \{a\}^*(\{a\}X_1 \cup \{b\}) \\
X_1 &= \{b,c\}^*(\{c\}X_2 \cup \{d\}) \\
X_2 &= \{c\}X_2 \cup (\{a\}\{a\}^+ \cup \{b\})(\{b,c\}^*(\{c\}X_2 \cup \{d\})) \cup \{a\}^+\{b\} \cup \{c,d\}) \\
&= \{c\}X_2 \cup (\{a\}\{a\}^+ \cup \{b\})(\{b,c\}^*\{c\}X_2 \cup \{b,c\}^*\{d\}) \cup \{a\}^+\{b\} \cup \{c,d\} \\
&= \{c\}X_2 \cup (\{a\}\{a\}^+ \cup \{b\})\{b,c\}^*\{c\}X_2 \cup (\{a\}\{a\}^+ \cup \{b\})\{b,c\}^*\{d\} \cup \\
&\quad \{a\}^+\{b\} \cup \{c,d\} \\
&= \{c\}X_2 \cup (\{a\}\{a\}^+ \cup \{b\})\{b,c\}^*\{c\}X_2 \cup \{a\}\{a\}^+\{b,c\}^*\{d\} \cup \\
&\quad \{b\}\{b,c\}^*\{d\} \cup \{a\}^+\{b\} \cup \{c,d\} \\
&= (\{c\} \cup (\{a\}\{a\}^+ \cup \{b\})\{b,c\}^*\{c\})X_2 \cup \{a\}\{a\}^+\{b,c\}^*\{d\} \cup \\
&\quad \{b\}\{b,c\}^*\{d\} \cup \{a\}^+\{b\} \cup \{c,d\}
\end{aligned}
$$

- Passos 1 e 2 do algoritmo, terceira passagem:

$$
\begin{aligned}
X_2 &= A_{22}^* C_2, \text{ com:} \\
A_{22} &= (\{c\} \cup (\{a\}\{a\}^+ \cup \{b\})\{b,c\}^*\{c\}) \\
C_2 &= \{a\}\{a\}^+\{b,c\}^*\{d\} \cup \{b\}\{b,c\}^*\{d\} \cup \{a\}^+\{b\} \cup \{c,d\}
\end{aligned}
$$

- Passos 4 e 5 do algoritmo, primeira passagem:
  Substituição de X_2 por $A_{22}^* C_2$ nas equações de X_0 e X_1:

$$
\begin{aligned}
X_0 &= \{a\}^*(\{a\}X_1 \cup \{b\}) \\
X_1 &= \{b,c\}^*(\{c\}(A_{22}^* C_2) \cup \{d\}) = \{b,c\}^*\{c\}A_{22}^* C_2 \cup \{b,c\}^*\{d\} \\
X_2 &= A_{22}^* C_2
\end{aligned}
$$

- Passos 4 e 5 do algoritmo, segunda passagem:
  Substituição de X_1 por $\{b,c\}^*\{c\}A_{22}^* C_2 \cup \{b,c\}^*\{d\}$ na equação de X_0:

$$
\begin{aligned}
X_0 &= \{a\}^*(\{a\}(\{b,c\}^*\{c\}A_{22}^* C_2 \cup \{b,c\}^*\{d\}) \cup \{b\}) \\
&= \{a\}^*(\{a\}\{b,c\}^*\{c\}A_{22}^* C_2 \cup \{a\}\{b,c\}^*\{d\} \cup \{b\}) \\
&= \{a\}^*\{a\}\{b,c\}^*\{c\}A_{22}^* C_2 \cup \{a\}^*\{a\}\{b,c\}^*\{d\} \cup \{a\}^*\{b\} \\
X_1 &= \{b,c\}^*\{c\}A_{22}^* C_2 \cup \{b,c\}^*\{d\} \\
X_2 &= A_{22}^* C_2
\end{aligned}
$$

Portanto, o conjunto regular definido por G_0 é $\Omega_0\Omega_1 \cup \Omega_3 \cup \Omega_4$, com:

$$
\begin{aligned}
\Omega_0 &= \{a\}^*\{a\}\{b,c\}^*\{c\}(\{c\} \cup (\{a\}\{a\}^+ \cup \{b\})\{b,c\}^*\{c\})^* \\
\Omega_1 &= \{a\}\{a\}^+\{b,c\}^*\{d\} \cup \{b\}\{b,c\}^*\{d\} \cup \{a\}^+\{b\} \cup \{c,d\} \\
\Omega_3 &= \{a\}^*\{a\}\{b,c\}^*\{d\} \\
\Omega_4 &= \{a\}^*\{b\}
\end{aligned}
$$

□

Exemplo 3.30 Seja G_1 uma gramática linear à direita:

$$\begin{aligned} G_1 &= (\{0,1,2,X_1,X_2,X_3\},\{0,1,2\},P_1,X_1) \\ P_1 &= \{X_1 \to 0X_1, X_1 \to X_2, \\ &\quad X_2 \to 1X_2, X_2 \to 1X_3, \\ &\quad X_3 \to 2X_3, X_3 \to 22\} \end{aligned}$$

Efetuando-se a mudança de notação e agrupando-se as alternativas, obtemos o sistema de equações:

$$\begin{aligned} X_1 &= \{0\}X_1 \cup \{\epsilon\}X_2 \\ X_2 &= \{1\}X_2 \cup \{1\}X_3 \\ X_3 &= \{2\}X_3 \cup \{22\} \end{aligned}$$

Pelo fato de esta gramática não conter referências à variável (não-terminal) X_1 nas equações de X_2 e X_3, nem tampouco referências à variável X_2 na equação da variável X_3, torna-se prático em primeiro lugar resolver diretamente a equação referente à variável X_3:

$$X_3 = \{2\}^*\{22\}$$

A seguir, substituem-se pela expressão acima todas as ocorrências da variável X_3 nas equações anteriores. O sistema de equações torna-se então:

$$\begin{aligned} X_1 &= \{0\}X_1 \cup \{\epsilon\}X_2 \\ X_2 &= \{1\}X_2 \cup \{1\}\{2\}^*\{22\} \\ X_3 &= \{2\}^*\{22\} \end{aligned}$$

De forma análoga, obtém-se $X_2 = \{1\}^*\{1\}\{2\}^*\{22\}$ e efetua-se a substituição desta expressão em todas as ocorrências da variável X_2 nas equações anteriores, que então se tornam:

$$\begin{aligned} X_1 &= \{0\}X_1 \cup \{\epsilon\}\{1\}^*\{1\}\{2\}^*\{22\} \\ X_2 &= \{1\}^*\{1\}\{2\}^*\{22\} \\ X_3 &= \{2\}^*\{22\} \end{aligned}$$

Finalmente, obtém-se $X_1 = \{0\}^*\{1\}^*\{1\}\{2\}^*\{22\}$, que representa também o conjunto regular definido pela gramática linear à direita G_1. Resulta:

$$\begin{aligned} X_1 &= \{0\}^*\{1\}^*\{1\}\{2\}^*\{22\} \\ X_2 &= \{1\}^*\{1\}\{2\}^*\{22\} \\ X_3 &= \{2\}^*\{22\} \end{aligned}$$

□

Exemplo 3.31 Seja G_2 uma gramática linear à direita:

$$\begin{aligned} G_2 &= (\{\sigma_0,\sigma_1,\sigma_2,A_0,A_1,A_2\},\{\sigma_0,\sigma_1\sigma_2\},P_2,A_0) \\ P2 &= \{A_0 \to \sigma_0 A_1, A_0 \to \mu_0, \\ &\quad A_1 \to \sigma_1 A_2, A_1 \to \mu_1, \\ &\quad A_2 \to \sigma_2 A_0, A_2 \to \mu_2\} \end{aligned}$$

Através de manipulações, obtém-se o sistema:

$$\begin{aligned} A_0 &= \{\sigma_0\}A_1 \cup \{\mu_0\} \\ A_1 &= \{\sigma_1\}A_2 \cup \{\mu_1\} \\ A_2 &= \{\sigma_2\}A_0 \cup \{\mu_2\} \end{aligned}$$

Substituindo-se todas as ocorrências de A_0 nas demais equações por $\{\sigma_0\}A_1 \cup \{\mu_0\}$:

$$\begin{aligned} A_0 &= \{\sigma_0\}A_1 \cup \{\mu_0\} \\ A_1 &= \{\sigma_1\}A_2 \cup \{\mu_1\} \\ A_2 &= \{\sigma_2\}(\{\sigma_0\}A_1 \cup \{\mu_0\}) \cup \{\mu_2\} \end{aligned}$$

Substituindo-se todas as ocorrências de A_1 nas demais equações por $\{\sigma_1\}A_2 \cup \{\mu_1\}$:

$$\begin{aligned} A_0 &= \{\sigma_0\}A_1 \cup \{\mu_0\} \\ A_1 &= \{\sigma_1\}A_2 \cup \{\mu_1\} \\ A_2 &= \{\sigma_2\}(\{\sigma_0\}(\{\sigma_1\}A_2 \cup \{\mu_1\}) \cup \{\mu_0\}) \cup \{\mu_2\} \end{aligned}$$

Reescrevendo-se a equação referente à variável A_2, tem-se:

$$\begin{aligned} A_2 &= \{\sigma_2\sigma_0\sigma_1\}A_2 \cup \{\sigma_2\sigma_0\,\mu_1, \sigma_2\,\mu_0, \mu_2\} \\ &= \{\sigma_2\sigma_0\sigma_1\}^*\{\sigma_2\sigma_0\,\mu_1, \sigma_2\,\mu_0, \mu_2\} \end{aligned}$$

Substituindo-se essa expressão na equação referente à variável A_1, tem-se:

$$A_1 = \{\sigma_1\}\{\sigma_2\sigma_0\sigma_1\}^*\{\sigma_2\sigma_0\,\mu_1, \sigma_2\,\mu_0, \mu_2\} \cup \{\mu_1\}$$

Finalmente, substituindo-se esta expressão na equação referente à variável A_0, vem:

$$\begin{aligned} A_0 &= \{\sigma_0\}(\{\sigma_1\}\{\sigma_2\sigma_0\sigma_1\}^*\{\sigma_2\sigma_0\,\mu_1, \sigma_2\,\mu_0, \mu_2\} \cup \mu_1\}) \cup \{\mu_0\} \\ &= \{\sigma_0\sigma_1\}\{\sigma_2\sigma_0\sigma_1\}^*\{\sigma_2\sigma_0\,\mu_1, \sigma_2\,\mu_0, \mu_2\} \cup \sigma_0\,\mu_1\} \cup \{\mu_0\} \end{aligned}$$

□

Exemplo 3.32 Seja G_3 uma gramática linear à direita:

$$\begin{aligned} G_3 &= (\{0, 1, 2, S, A\}, \{0, 1, 2\}, P_3, S) \\ P_3 &= \{S \to 0A, S \to 1, A \to 0S, A \to 2\} \end{aligned}$$

Os passos seguintes levam à caracterização do conjunto regular definido por G_3:

$$\begin{aligned} S &= \{0\}A \cup \{1\} \\ A &= \{0\}S \cup \{2\} \end{aligned}$$

Substituindo-se a definição de S na equação de A, vem:

$$\begin{aligned} A &= \{0\}(\{0\}A \cup \{1\}) \cup \{2\} \\ &= \{00\}A \cup \{01\} \cup \{2\} \\ &= \{00\}^*\{01, 2\} \end{aligned}$$

Substituindo-se todas as ocorrências de A pela expressão acima na equação de S, tem-se:

$$\begin{aligned} S &= \{0\}(\{00\}^*\{01, 2\}) \cup \{1\} \\ &= \{0\}\{00\}^*\{01, 2\} \cup \{1\} \\ &= \{00\}^*\{02, 1\} \end{aligned}$$

□

3.5 Equivalência entre Gramáticas Regulares e Autômatos Finitos

Dando seqüência ao conteúdo cuja discussão foi iniciada na Seção 3.4, é mostrada a seguir a equivalência entre as gramáticas lineares à direita e os autômatos finitos, no que diz respeito à classe de linguagens que tais dispositivos são capazes de definir. Finalmente, e embora redundante do ponto de vista teórico (pois a equivalência dos conjuntos regulares com os autômatos finitos pode ser verificada indiretamente através das gramáticas lineares à direita), é apresentada uma forma direta de grande interesse prático para a obtenção de autômatos finitos a partir de expressões regulares.

As equivalências discutidas nesta seção estão representadas com destaque na Figura 3.23.

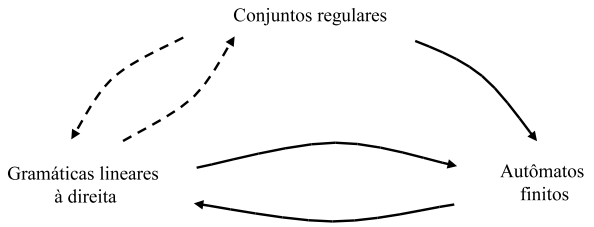

Figura 3.23: Equivalência dos formalismos — parte 2

Teorema 3.9 (Gramáticas lineares à direita ⇒ autômatos finitos) *Seja G uma gramática linear à direita. Então é possível definir um autômato finito M de tal modo que $L(G) = L(M)$.*

Justificativa Considere-se $G = (V, \Sigma, P, S)$. As produções de P são do tipo:

1. $X \rightarrow aY$
2. $X \rightarrow Y$
3. $X \rightarrow a$
4. $X \rightarrow \epsilon$

com $X, Y \in N$, $a \in \Sigma$.

Seja $M = (Q, \Sigma, \delta, q_0, F)$ o autômato que se deseja obter. O Algoritmo 3.14 mostra como construir M (não-determinístico, com transições em vazio) a partir de G:

Algoritmo 3.14 (Gramática ⇒ autômato) *Construção de um autômato finito a partir de uma gramática linear à direita.*

- Entrada: uma gramática linear à direita G;
- Saída: um autômato finito M tal que $L(M) = L(G)$;
- Método:

1. Conjunto de estados:

   Cada estado de M corresponde a um dos símbolos não-terminais de G. A esse conjunto acrescenta-se um novo símbolo (estado) $Z \notin N$, ou seja, $Q = N \cup \{Z\}$. O estado inicial de M é S, a raiz da gramática. O estado final de M é Z, o novo estado acrescentado.

2. Alfabeto de entrada:

   O alfabeto de entrada Σ de M é o mesmo alfabeto Σ de G.

3. Função de transição:

   $\delta \leftarrow \varnothing$;

   Para cada regra de produção em P da gramática G, e conforme seu tipo:

   a) Se $X \rightarrow aY$ então $\delta \leftarrow \delta \cup \{(X, a) \rightarrow Y\}$;

   b) Se $X \rightarrow Y$ então $\delta \leftarrow \delta \cup \{(X, \epsilon) \rightarrow Y\}$;

   c) Se $X \rightarrow a$ então $\delta \leftarrow \delta \cup \{(X, a) \rightarrow Z\}$;

   d) Se $X \rightarrow \epsilon$ então $\delta \leftarrow \delta \cup \{(X, \epsilon) \rightarrow Z\}$.

Note-se que, pela aplicação do Algoritmo 3.14, $\big|Q\big| = \big|N\big| + 1$ e $\big|\delta\big| = \big|P\big|$. ■

O algoritmo acima apresentado pode ser melhor compreendido considerando-se que não apenas a raiz S da gramática, mas também cada não-terminal X dessa gramática gera, por si só, uma linguagem:

$$L(X) = \{\alpha \in \Sigma^* \mid X \Rightarrow^* \alpha\}$$

Assim, por exemplo, produções do tipo $X \rightarrow aY$ podem ser interpretadas como "a linguagem gerada pelo não-terminal X corresponde ao símbolo a concatenado à linguagem gerada pelo não-terminal Y". Situação semelhante ocorre com os autômatos finitos: não apenas o estado inicial, mas qualquer estado X também define uma linguagem:

$$L(X) = \{\alpha \in \Sigma^* \mid (X, \alpha) \vdash^* (Y, \epsilon), Y \in F\}$$

Dessa maneira, ao transformar símbolos não-terminais em estados, a produção correspondente $(S, a) \rightarrow Y$ adota o significado "a linguagem aceita pelo estado X corresponde ao símbolo a seguido da linguagem aceita pelo estado Y", o que sugere a validade do algoritmo proposto quando da aplicação desse raciocínio à raiz da gramática, transformada em estado inicial do autômato correspondente, bem como para quaisquer outros não-terminais e respectivos estados.

Novamente, cabe frisar que todas as linguagens assim definidas a partir dos diversos não-terminais da gramática são linguagens geradas por gramáticas lineares à direita (definidas por subconjuntos do conjunto de produções P original).

Exemplo 3.33 Seja G uma gramática linear à direita:

$$
\begin{aligned}
G &= (V, \Sigma, P, S) \\
V &= \{a, b, c, S, K, L\} \\
\Sigma &= \{a, b, c\} \\
P &= \{S \rightarrow a, S \rightarrow aK, K \rightarrow bK, K \rightarrow L, L \rightarrow cL, L \rightarrow \epsilon\}
\end{aligned}
$$

Aplicando-se o algoritmo de conversão a G, obtém-se o autômato da Tabela 3.39.

P	δ
$S \rightarrow a$	$\delta(S, a) = Z$
$S \rightarrow aK$	$\delta(S, a) = K$
$K \rightarrow bK$	$\delta(K, b) = K$
$K \rightarrow L$	$\delta(K, \epsilon) = L$
$L \rightarrow cL$	$\delta(L, c) = L$
$L \rightarrow \epsilon$	$\delta(L, \epsilon) = Z$

Tabela 3.39: Mapeamento das produções da gramática G para as regras de transição do autômato M

O autômato finito completo M assim especificado é apresentado na Figura 3.24. Note-se que $L(G) = L(M) = ab^*c^*$.

$$
\begin{aligned}
M &= (Q, \Sigma, \delta, S, F) \\
Q &= \{S, K, L, Z\} \\
\Sigma &= \{a, b, c\} \\
\delta &= \{(S, a) \rightarrow Z, (S, a) \rightarrow K, (K, b) \rightarrow K, (K, \epsilon) \rightarrow L, (L, c) \rightarrow L, (L, \epsilon) \rightarrow Z\} \\
F &= \{Z\}
\end{aligned}
$$

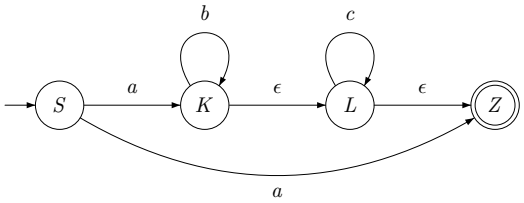

Figura 3.24: Autômato resultante do mapeamento mostrado na Tabela 3.39

Considere-se a sentença $abbcc$, e as correspondentes seqüências de derivações em G e de movimentações em M:

- $S \Rightarrow aK \Rightarrow abK \Rightarrow abbK \Rightarrow abbL \Rightarrow abbcL \Rightarrow abbccL \Rightarrow abbcc$
- $(S, abbcc) \vdash (K, bbcc) \vdash (K, bcc) \vdash (K, cc) \vdash (L, cc) \vdash (L, c) \vdash (L, \epsilon) \vdash (Z, \epsilon)$

Comparando-se as formas sentenciais geradas por G com as configurações assumidas por M, é fácil perceber que há uma relação direta entre elas, expressa na idéia anteriormente exposta de que a linguagem gerada por um certo símbolo não-terminal de G corresponde à linguagem reconhecida pelo respectivo estado de M. Note-se, em particular, que o número de formas sentenciais geradas por G é igual ao número de configurações assumidas por M no reconhecimento da mesma cadeia de entrada (no caso, oito formas sentenciais e oito configurações). □

Teorema 3.10 (Gramáticas lineares à direita ⇐ autômatos finitos) *Seja M um autômato finito qualquer. Então é possível definir uma gramática linear à direita G, de tal modo que $L(M) = L(G)$.*

Justificativa Considere-se $M = (Q, \Sigma, \delta, q_0, F)$ um autômato finito genérico, não-determinístico e com transições em vazio. O Algoritmo 3.15 mostra como construir uma gramática linear à direita $G = (V, \Sigma, P, S)$ a partir de M.

Algoritmo 3.15 (Gramática ⇐ autômato) *Construção de uma gramática linear à direita a partir de um autômato finito.*

- Entrada: um autômato finito M;
- Saída: uma gramática linear à direita G tal que $L(G) = L(M)$;
- Método:

  1. Definição do conjunto de símbolos não-terminais:

     Os símbolos não-terminais de G correspondem aos estados de M. A raiz da gramática é q_0.

  2. Alfabeto de entrada:

     O alfabeto Σ de G é o próprio alfabeto de entrada Σ de M.

  3. Produções:

     $P \leftarrow \varnothing$;

     Para cada elemento de δ da máquina M, e conforme o tipo das transições de M:

     a) Se $\delta(X, a) = Y$, então $P \leftarrow \{X \rightarrow aY\}$;

     b) Se $\delta(X, \epsilon) = Y$, então $P \leftarrow \{X \rightarrow Y\}$.

     $F \leftarrow \varnothing$

     Para cada elemento de Q da máquina M:

     a) Se $X \in F$, então $P \leftarrow \{X \rightarrow \epsilon\}$.

Note-se que, pela aplicação do Algoritmo 3.15, $\big|N\big| = \big|Q\big|$ e $\big|P\big| = \big|\delta \ \big| + \big| \ F\big|$. O funcionamento desse algoritmo de mapeamento é simples e pode ser compreendido através de considerações similares às efetuadas no Teorema 3.9. ■

Exemplo 3.34 Seja M representado na Figura 3.25:

$$\begin{aligned} M &= (Q, \Sigma, \delta, q_0, F) \\ Q &= \{q_0, q_1, q_2\} \\ \Sigma &= \{a, b, c\} \end{aligned}$$

$$\begin{aligned} \delta &= \{(q_0, a) \rightarrow q_1, (q_1, b) \rightarrow q_1, (q_1, c) \rightarrow q_2, (q_1, \epsilon) \rightarrow q_2, (q_2, c) \rightarrow q_2\} \\ F &= \{q_2\} \end{aligned}$$

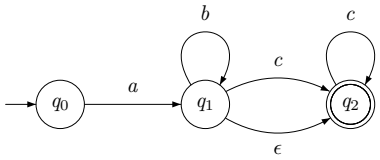

Figura 3.25: Autômato original M do Exemplo 3.34

Aplicando-se o algoritmo de conversão à máquina M, obtém-se a gramática linear à direita G apresentada na Tabela 3.40, cujo conjunto de produções P corresponde à segunda coluna da mesma. Note que $L(M) = L(G) = ab^*c^*$.

$$\begin{aligned} G &= (V, \Sigma, P, q_0) \\ V &= \{a, b, c, q_0, q_1, q_2\} \\ \Sigma &= \{a, b, c\} \end{aligned}$$

δ	P
$\delta(q_0, a) = q_1$	$q_0 \rightarrow aq_1$
$\delta(q_1, b) = q_1$	$q_1 \rightarrow bq_1$
$\delta(q_1, c) = q_2$	$q_1 \rightarrow cq_2$
$\delta(q_1, \epsilon) = q_2$	$q_1 \rightarrow q_2$
$\delta(q_2, c) = q_2$	$q_2 \rightarrow cq_2$
Q	P
$q_2 \in F$	$q_2 \rightarrow \epsilon$

Tabela 3.40: Gramática G equivalente ao autômato M da Figura 3.25

Considere-se a cadeia $abbbc$, e as correspondentes seqüências de movimentações em M e de derivações em G:

- $(q_0, abbbc) \vdash (q_1, bbbc) \vdash (q_1, bbc) \vdash (q_1, bc) \vdash (q_1, c) \vdash (q_2, \epsilon)$
- $q_0 \Rightarrow aq_1 \Rightarrow abq_1 \Rightarrow abbq_1 \Rightarrow abbbq_1 \Rightarrow abbbcq_2 \Rightarrow abbbc$

Em particular, a cadeia $abbbc$ possui mais de uma seqüência de movimentações que conduzem à sua aceitação em M. Tal fato implica a existência de uma outra seqüência de derivações que é capaz de gerar essa cadeia em G, como pode ser percebido abaixo:

- $(q_0, abbbc) \vdash (q_1, bbbc) \vdash (q_1, bbc) \vdash (q_1, bc) \vdash (q_1, c) \vdash (q_2, c) \vdash (q_2, \epsilon)$
- $q_0 \Rightarrow aq_1 \Rightarrow abq_1 \Rightarrow abbq_1 \Rightarrow abbbq_1 \Rightarrow abbbq_2 \Rightarrow abbbcq_2 \Rightarrow abbbc$

Assim como no caso do Exemplo 3.33, a comparação entre as configurações assumidas por M e as formas sentenciais geradas por G revela que há uma relação direta entre elas, expressa na mesma idéia de que a linguagem reconhecida por um certo estado de M corresponde à linguagem gerada pelo respectivo símbolo não-terminal de G. Note-se, em particular, que o número de formas sentenciais obtidas por G na geração da cadeia é igual ao número de configurações assumidas por M

no reconhecimento da mesma cadeia, porém acrescido de um (no primeiro caso seis configurações e sete formas sentenciais, e no segundo caso sete configurações e oito formas sentenciais). □

Os resultados apresentados até este ponto são suficientes para estabelecer intuitivamente a equivalência completa dos conjuntos (expressões) regulares com as gramáticas lineares à direita e também com os autômatos finitos, no que se refere à classe de linguagens que tais dispositivos são capazes de representar.

Não obstante, a fim de possibilitar uma visualização mais completa da relação de equivalência entre os autômatos finitos e as expressões regulares, é apresentado em seguida um algoritmo que permite a construção sistemática de autômatos finitos não-determinísticos diretamente a partir de expressões regulares quaisquer.

Teorema 3.11 (Expressões regulares ⇒ autômatos finitos) *Seja r uma expressão regular sobre o alfabeto Σ. Então existe um autômato finito M que aceita a linguagem definida por r.*

Justificativa O autômato finito não-determinístico que aceita a linguagem definida por r pode ser obtido através da aplicação do Algoritmo 3.16, que especifica as regras de mapeamento parciais que abrangem casos triviais de sentenças (itens 1, 2 e 3) e cada um dos operadores de união (4), concatenação (5) e fechamento (6), conforme a própria definição das expressões regulares.

Algoritmo 3.16 (Expressão regular ⇒ autômato) *Obtenção de um autômato finito a partir de uma expressão regular.*

- Entrada: uma expressão regular r sobre um alfabeto Σ;
- Saída: um autômato finito M tal que $L(M) = r$;
- Método:

1. $r = \epsilon$

   r é aceita por:

Figura 3.26: Autômato que aceita ϵ

2. $r = \varnothing$

   r é aceita por:

Figura 3.27: Autômato que aceita $\varnothing$

3. $r = \sigma, \sigma \in \Sigma$

   r é aceita por:

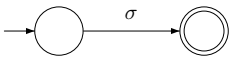

Figura 3.28: Autômato que aceita σ

4. $r_3 = r_1 \mid r_2$, com $L(M_1) = r_1$ e $L(M_2) = r_2$

   r_3 é aceita por:

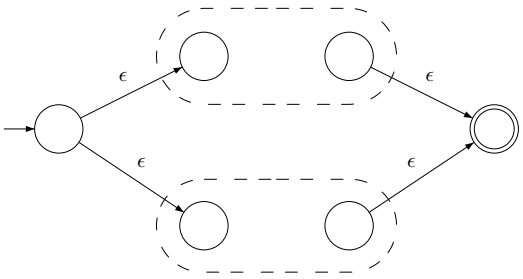

Figura 3.29: Autômato que aceita $r_1 \mid r_2$

As cadeias aceitas por M_3 correspondem àquelas que são aceitas por M_1 e também àquelas que são aceitas por M_2. Logo, $L(M_3) = L(M_1) \cup L(M_2)$.

5. $r_3 = r_1 r_2$, com $L(M_1) = r_1$ e $L(M_2) = r_2$

   r_3 é aceita por:

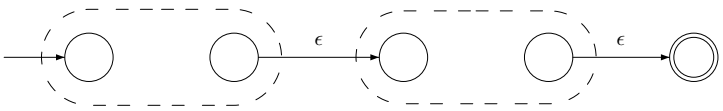

Figura 3.30: Autômato que aceita $r_1 r_2$

As cadeias aceitas por M_3 correspondem àquelas cujos prefixos sejam cadeias aceitas por M_1 e cujos sufixos sejam cadeias que sejas aceitas por M_2. Logo, $L(M_3) = L(M_1)L(M_2)$.

6. $r_3 = r_1^*$, com $L(M_1) = r_1$

r_3 é aceita por:

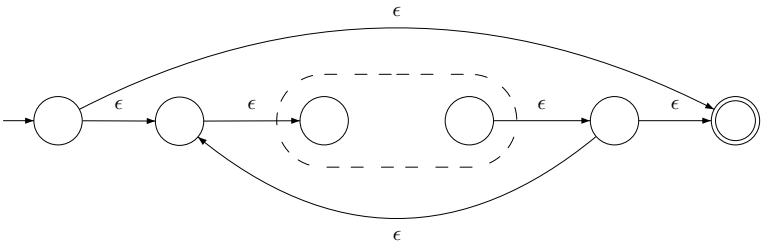

Figura 3.31: Autômato que aceita r_1^*

M_3 aceita a cadeia vazia e também a concatenação, em número arbitrário de vezes, de quaisquer cadeias que são aceitas por M_1. Logo, $L(M_3) = L(M_1)^*$.

Adicionalmente, valem as seguintes regras para os autômatos M_1 e M_2 representados nos casos (4), (5) e (6):

- Os estados originalmente finais desses autômatos tornam-se não-finais na composição e se comportam apenas como estados de saída do autômato;
- Convenciona-se que a entrada dos mesmos é pelo lado esquerdo e a saída pelo lado direito das respectivas figuras.

■

Exemplo 3.35 Considere-se a expressão regular $ab^* \mid c$. É possível identificar, nessa expressão, as seguintes linguagens triviais: $L_1 = a, L_2 = b, L_3 = c$. Portanto, $L_1 = L_1(M_1), M_1 =$

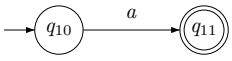

Figura 3.32: Autômato que aceita a

$L_2 = L_2(M_2), M_2 =$

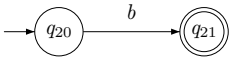

Figura 3.33: Autômato que aceita b

$L_3 = L_3(M_3), M_3 =$

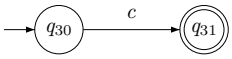

Figura 3.34: Autômato que aceita c

Seja $L_4 = b^* = L_2^*$. Então, $L_4 = L_4(M_4)$ com M_4 igual a:

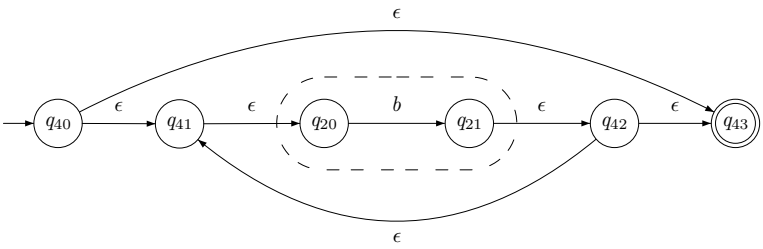

Figura 3.35: Autômato que aceita b^*

$L_5 = ab^* = L_1 L_4$. Então, $L_5 = L_5(M_5)$ com M_5 igual a:

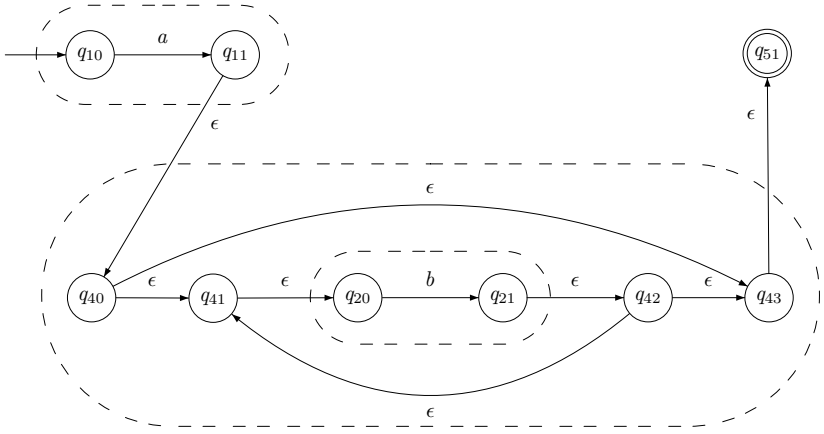

Figura 3.36: Autômato que aceita ab^*

Finalmente, $L_6 = ab^* \mid c = L_5 \cup L_3 = L_6(M_6)$ com M_6 igual a:

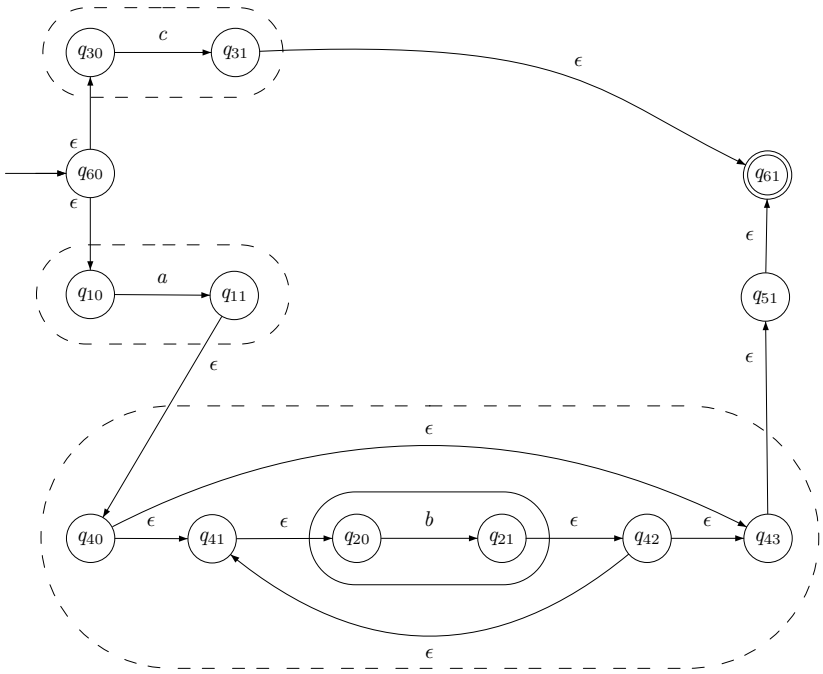

Figura 3.37: Autômato que aceita $ab^* \mid c$

Como se pode observar, o autômato assim construído apresenta uma quantidade desnecessariamente grande de estados, além de diversas transições em vazio. Este fenômeno, comum quando se aplicam métodos canônicos de construção de autômatos, pode ser contornado através da aplicação dos algoritmos de eliminação de transições em vazio e de estados inacessíveis, além da aplicação de algoritmos de minimização, assunto da Seção 3.6. O autômato da Figura 3.38 corresponde a uma versão equivalente, porém mais compacta, ao autômato M_6 anteriormente obtido.

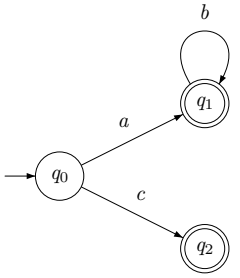

Figura 3.38: Outro autômato que aceita $ab^* \mid c$

□

É comum se considerar uma simplificação do autômato apresentado na Figura 3.31, no qual, ao invés de quatro novos estados, são criados apenas dois novos estados, como ilustrado na Figura 3.39.

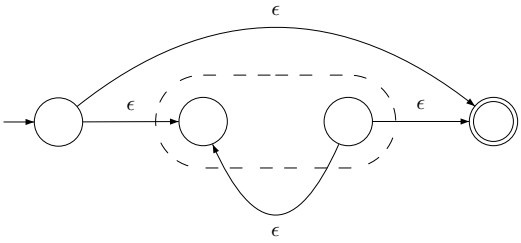

Figura 3.39: Autômato alternativo que aceita r_1^*

Assim como no caso dos autômatos apresentados nas Figuras 3.29 e 3.30, o autômato da Figura 3.31 preserva as funções de transição dos autômatos originais inalteradas. O autômato da Figura 3.39, por outro lado, exige o acréscimo de uma transição em vazio do (então) estado final para o (então) estado inicial do autômato original, porém produz resultados equivalentes aos obtidos com o autômato da Figura 3.31.

Exemplo 3.36
O autômato da Figura 3.40 reconhece a linguagem gerada pela expressão regular a^*b^*.

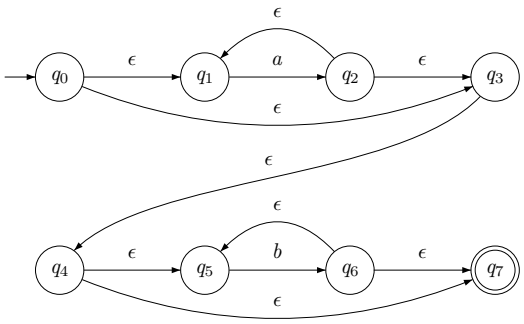

Figura 3.40: Autômato que reconhece a^*b^*

□

Para finalizar, a Tabela 3.41 apresenta um resumo que oferece uma visão abrangente das várias possibilidades de conversão direta entre os diversos formalismos estudados para representar as linguagens regulares, conforme discutido nas Seções 3.1 a 3.5.

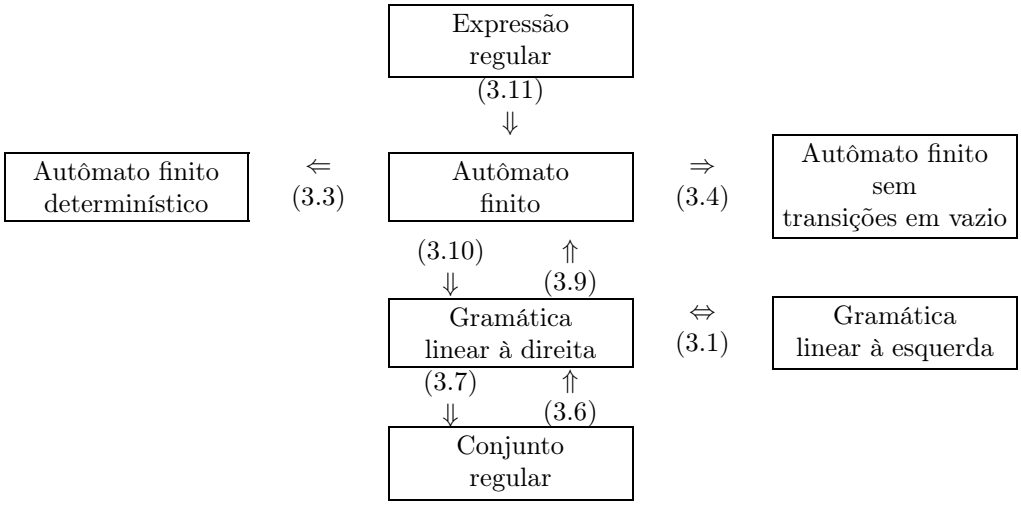

Tabela 3.41: Visão geral da equivalência dos formalismos e respectivos teoremas

3.6 Minimização de Autômatos Finitos

Um importante resultado da teoria dos autômatos refere-se à equivalência da classe dos autômatos finitos determinísticos com a dos não-determinísticos, ou seja, não-determinismos em autômatos finitos em nada contribuem para ampliar a classe de linguagens por eles reconhecíveis. Conforme mostrado anteriormente, é sempre possível transformar autômatos finitos não-determinísticos que contenham transições em vazio em outros equivalentes, determinísticos, isentos de tais transições.

Nesta seção é apresentado talvez o mais importante resultado teórico conhecido para a classe das linguagens regulares: o da existência de autômatos finitos determinísticos, mínimos e únicos, que reconhecem os respectivos conjuntos regulares. Em outras palavras, pode-se provar que cada conjunto regular é reconhecido por um autômato finito mínimo e único. O termo **mínimo** é empregado para designar um autômato finito que tenha o número mínimo possível de estados.

A importância desse resultado decorre de uma série de fatores. Em primeiro lugar, porque foi demonstrado que ele é válido apenas para a classe das linguagens definidas por autômatos finitos, não havendo correspondente para outras classes de linguagens; em segundo lugar, porque ele extrapola o interesse puramente teórico, despertando muito interesse prático, já que possibilita a construção de reconhecedores sintáticos extremamente compactos e eficientes; em terceiro lugar, porque é possível automatizar a minimização de autômatos finitos, ou seja, existe um algoritmo que é capaz de transformar qualquer autômato finito em uma versão equivalente mínima; e, finalmente, em quarto lugar, porque o autômato finito mínimo é único para cada linguagem regular, e isso possibilita a elaboração de novos métodos no estudo de linguagens formais, como, por exemplo, a demonstração da equivalência de duas linguagens através da redução dos correspondentes autômatos finitos às suas versões equivalentes mínimas.

O método apresentado neste item parte da hipótese de que o autômato a ser minimizado é determinístico e portanto, obviamente, isento de transições em vazio. Dado um autômato finito qualquer, a aplicação dos algoritmos apresentados na Seção 3.3 permite

a sua transformação em autômatos equivalentes sem transições em vazio ou qualquer outro tipo de não-determinismo.

A minimização do número de estados de um autômato finito é feita em duas etapas:

1. Eliminação de estados inacessíveis e inúteis;

2. Agrupamento e fusão de estados equivalentes.

Estados inacessíveis e inúteis podem ser eliminados através da aplicação dos algoritmos apresentados anteriormente, ficando então o autômato finito determinístico apto a ser reduzido à sua versão mínima através da fusão dos seus estados equivalentes.

O princípio desse método consiste em se efetuar o particionamento sistemático do conjunto de estados do autômato em grupos de estados sucessivamente mais abrangentes, até que cada grupo contenha o maior número de estados possível, o que caracteriza a respectiva **classe de equivalência**. Uma vez esgotado o processo, ou seja, quando não mais for possível modificar os grupos construídos, escolhe-se um representante único para cada classe, descartando-se os demais estados da mesma classe, uma vez que se trata de estados equivalentes.

A seguir serão apresentadas algumas definições e os métodos que permitem efetuar a identificação sistemática das classes de equivalência de um autômato finito determinístico M, bem como a construção do autômato finito mínimo equivalente M' a partir de M.

Considerem-se $M = (Q, \Sigma, \delta, q_0, F)$, um autômato finito determinístico, e dois de seus estados, $q_1, q_2 \in Q$. Diz-se que a cadeia $x \in \Sigma^*$ **distingue** q_1 de q_2 se $(q_1, x) \vdash^* (q_3, \epsilon), (q_2, x) \vdash^* (q_4, \epsilon)$ e, de forma exclusiva, ou $q_3 \in F$ ou então $q_4 \in F$. Em outras palavras, uma cadeia x distingue q_1 de q_2, quando, para ser integralmente consumida a partir de cada um desses dois estados, ela conduzir o autômato a um estado final em apenas um desses casos.

Dois estados q_1 e q_2 são ditos k-**indistinguíveis** (denotado por $q_1 \overset{k}{\equiv} q_2$) se e apenas se não houver cadeia $x, \left|x\right| \leq k$, que permita distinguir q_1 de q_2. Finalmente, q_1 e q_2 são ditos **indistinguíveis**, ou **equivalentes** (denotado por $q_1 \equiv q_2$), se e apenas se eles forem k-indistinguíveis para todo $k \geq 0$.

Observe-se que, de acordo com a definição, para qualquer par de estados $q_i, q_j \in Q$, valem as duas seguintes condições:

1. $q_i \overset{0}{\equiv} q_j$ se e apenas se ou $q_i, q_j \in F$, ou então $q_i, q_j \in (Q - F)$, e

2. $q_i \overset{k}{\equiv} q_j$ se e apenas se $q_i \overset{k-1}{\equiv} q_j$ e $\delta(q_i, \sigma) \overset{k-1}{\equiv} \delta(q_j, \sigma), \forall\, \sigma \in \Sigma$.

A relação "$\equiv$" possibilita o agrupamento dos estados de um autômato finito M através da obtenção das denominadas classes de equivalência, em que os estados pertencentes a cada classe são todos equivalentes (indistinguíveis) entre si. Uma vez agrupados os estados equivalentes, basta escolher um único representante para cada classe, substituir por este todas as referências aos demais estados da mesma classe, e eliminar todos estes, garantindo-se assim a total ausência de redundâncias de estados da versão final de M.

No entanto, a identificação de pares de estados indistinguíveis requer, conforme visto acima, a indistinguibilidade para cadeias de comprimento qualquer. Naturalmente, a verificação exaustiva da equivalência de estados para cadeias de comprimento arbitrário é um sério limitante para a elaboração de um algoritmo que possa automatizar essa operação. Por tal motivo, o Teorema 3.12 é fundamental como base para permitir a

realização prática do método de minimização de estados pela identificação das classes de equivalência.

Teorema 3.12 (Estados equivalentes) *Seja $M = (Q, \Sigma, \delta, q_0, F)$ um autômato finito determinístico com n estados, e considere dois estados quaisquer $q_1, q_2 \in Q$. Então, $q_1 \equiv q_2$ se e apenas se $q_1 \overset{n-2}{\equiv} q_2$.*

Justificativa Este teorema afirma que, para se garantir a equivalência de dois estados em um autômato finito com n estados, é suficiente garantir a sua $(n-2)$-indistinguibilidade, ou seja, não há necessidade de se considerar cadeias de comprimento maior que $n-2$.

A demonstração da condição suficiente é trivial, uma vez que, por definição, se um par de estados é k_1-indistinguível, isso implica que ele é também k_2-indistinguível, $k_2 \leq k_1$.

A demonstração da condição necessária, ou seja, a $(n-2)$-indistinguibilidade implica indistinguibilidade total, é trivial também no caso em que os n estados do autômato finito são todos estados finais ou mesmo não-finais. Considere-se, portanto, o caso mais geral em que há tanto estados finais como estados não-finais em M.

De acordo com o critério $\overset{0}{\equiv}$, o conjunto Q pode ser particionado inicialmente em dois grandes subconjuntos: o primeiro, formado por todos os estados finais (F), e o segundo, por todos os estados não-finais ($Q - F$). Trata-se, portanto, do primeiro de uma série de sucessivos refinamentos com o objetivo de determinar as classes de equivalência de estados no autômato M.

Executa-se em seguida, para cada um dos dois subconjuntos obtidos através de $\overset{0}{\equiv}$, o seu refinamento (particionamento) através de relações $\overset{i}{\equiv}, i = 1, 2, 3$ etc. Como M possui n estados, o maior subconjunto de Q criado através de $\overset{0}{\equiv}$ contém no máximo $n-1$ estados. Admitindo-se o pior caso, ou seja, que cada um desses estados constitua individualmente uma classe de equivalência distinta das demais, então haverá no máximo mais $n-2$ refinamentos sucessivos de $\overset{0}{\equiv}$ gerando conjuntos de classes de equivalência, distintas umas das outras.

Para completar a demonstração do teorema, basta provar, conforme descrito a seguir, que cada um desses $n-2$ particionamentos distintos sucessivos (no máximo) refere-se ao uso correspondente de cadeias de comprimento $1, 2, ..., n-2$, para efetuar o teste de distinguibilidade do par de estados. Conseqüentemente, não há possibilidade de ocorrer um novo particionamento distinto dos anteriores para cadeias de comprimento x se os particionamentos obtidos para cadeias de comprimento $x-1$ e $x-2$ se mostrarem idênticos. Dito ainda de outra maneira, se um certo particionamento feito sobre cadeias de comprimento x ainda não representa as classes de equivalência de M, então um novo particionamento deve necessariamente existir quando cadeias de comprimento $x+1$ forem consideradas.

Para provar essa afirmação, considere-se o conjunto de todas as classes de equivalência de M que satisfazem simultaneamente às relações $\overset{k}{\equiv}$ e $\overset{k+1}{\equiv}$. Não é difícil perceber, nesse caso, que essas mesmas classes de equivalência satisfazem a $\overset{k+2}{\equiv}$, $\overset{k+3}{\equiv}$ e assim sucessivamente.

Considere-se, por exemplo, uma situação hipotética em que (i) a relação $\overset{k}{\equiv}$ particiona um certo conjunto Q em três subconjuntos Q_0, Q_1 e Q_2; (ii) a relação $\overset{k+1}{\equiv}$ preserva o particionamento da relação $\overset{k}{\equiv}$ inalterado; e, finalmente, a relação $\overset{k+2}{\equiv}$ produz uma partição diferente, por exemplo, Q_0, R, S e Q_2, com $Q_1 = R \cup S$:

$$\begin{aligned} \overset{k}{\equiv} &: Q_0, Q_1, Q_2 \\ \overset{k+1}{\equiv} &: Q_0, Q_1, Q_2 \\ \overset{k+2}{\equiv} &: Q_0, R, S, Q_2 \end{aligned}$$

onde Q_0, Q_1, Q_2, R e S são conjuntos de estados, $Q = Q_0 \cup Q_1 \cup Q_2$, $Q_1 = R \cup S$ e $R \cap S = \varnothing$.

Admitindo-se, por hipótese, que Q_1 seja particionado em duas novas classes de equivalência R e S, isso significa que existe $q_i, q_j \in Q_1$, de modo que $q_i \overset{k+2}{\not\equiv} q_j$. Ora, para que isso fosse verdade, seria necessário, de acordo com a definição, que:

1. $q_i \overset{k+1}{\not\equiv} q_j$, ou

2. $\delta(q_i, \sigma) \overset{k+1}{\not\equiv} \delta(q_j, \sigma)$.

A condição (1) é falsa, pois, de acordo com a hipótese original, q_i e q_j pertencem ao conjunto Q_1, logo $q_i \overset{k+1}{\equiv} q_j$. A condição (2) também é falsa, pois, se $q_i \overset{k+1}{\equiv} q_j$, então $\delta(q_i, \sigma) \overset{k}{\equiv} \delta(q_j, \sigma)$. Como, por hipótese, as partições produzidas pelas relações $\overset{k}{\equiv}$ e $\overset{k+1}{\equiv}$ são idênticas, então $\delta(q_i, \sigma) \overset{k+1}{\equiv} \delta(q_j, \sigma)$.

Fica, portanto, demonstrado por contradição que, na hipótese de serem obtidos dois conjuntos idênticos de classes de equivalência para k e $k+1$, não haverá mais necessidade de se analisar a equivalência de tais classes para valores maiores do que k. Fica também demonstrado que, para um autômato finito com n estados, haverá no máximo $n-1$ conjuntos distintos de classes de equivalência ($\overset{0}{\equiv}$ e os demais $n-2$), cada qual respectivamente associado a cadeias de comprimento 0 até $n-2$, não havendo, portanto, necessidade de se examinar a equivalência de tais classes para cadeias de comprimento superior a $n-2$. ■

A seguir apresenta-se o Algoritmo 3.17, que permite efetuar o particionamento sistemático do conjunto Q de estados de um autômato finito, produzindo como resultado as classes de equivalências mais amplas existentes em Q.

Algoritmo 3.17 (Classes de equivalência) *Determinação das classes de equivalência existentes no conjunto Q de estados de um autômato finito.*

- Entrada: um autômato finito $M = (Q, \Sigma, \delta, q_0, F)$;
- Saída: uma partição $Q_0, Q_1, ..., Q_n$ do conjunto Q de estados, de tal forma que seus elementos correspondem às mais amplas classes de equivalências de estados existentes em Q.
- Método:

    1. Divide-se o conjunto original Q de estados de M nos dois subconjuntos que compõem a sua partição inicial: o primeiro, composto por todos os estados finais, e o outro, por todos os estados não-finais de Q. Tal ação se

justifica facilmente, já que um estado final, qualquer que seja ele, é sempre distinguível de um estado não-final, qualquer que seja ele (condição válida apenas para autômatos isentos de transições em vazio): a cadeia ϵ, aceita por um estado final e não aceita por um estado não-final, distingue um do outro. Essa partição inicial corresponde, portanto, ao resultado da aplicação da relação $\overset{0}{\equiv}$ ao conjunto Q.

2. Para cada um dos conjuntos obtidos em (1), refiná-los em novas partições, segundo o critério:

   Dois estados A e B de um mesmo subconjunto Q_i, obtido de uma partição prévia do conjunto Q de estados, são equivalentes se e somente se:

   a) A e B aceitam o mesmo conjunto S de entradas, $S \subseteq \Sigma$, e

   b) Para cada uma dessas entradas $\sigma \in S$:

      - $\delta(A, \sigma) = \delta(B, \sigma)$, ou
      - $\delta(A, \sigma) \neq \delta(B, \sigma)$, mas $\delta(A, \sigma)$ e $\delta(B, \sigma)$ são equivalentes.

   Caso contrário, eles não são equivalentes e devem, portanto, ensejar uma partição de Q_i.

Note-se, no Algoritmo 3.17, que a definição da equivalência entre dois estados A e B quaisquer é elaborada, inicialmente, a partir da identificação das entradas aceitas em cada um desses estados e, posteriormente, a partir da análise dos estados que são atingidos quando se transita com cada uma dessas entradas em A e B — se os estados atingidos são idênticos ou equivalentes. As Figuras 3.41, 3.42 e 3.43 ilustram tais situações para os estados A e B de um autômato que aceita duas entradas diferentes σ_1 e σ_2.

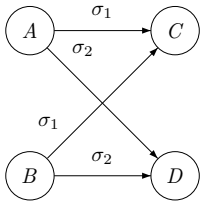

Figura 3.41: Transições com as mesmas entradas para estados idênticos

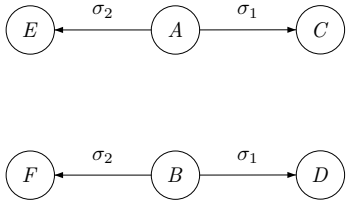

Figura 3.42: Transições com as mesmas entradas para estados equivalentes

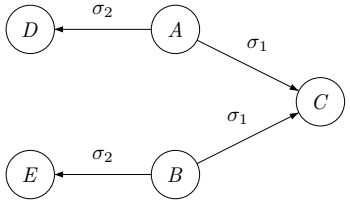

Figura 3.43: Transições com as mesmas entradas para estados idênticos e equivalentes

Exemplo 3.37 Considere-se o autômato da Tabela 3.42, similar ao autômato do Exemplo 3.38:

	δ	a	b	c	d	e
$\rightarrow$	q_0	q_1	q_3			
	q_1			q_3	q_1	q_2
(f)	q_2		q_6			
	q_3			q_1	q_3	q_4
(f)	q_4					
	q_5	q_4				
	q_6					

Tabela 3.42: Autômato original do Exemplo 3.37

Através da aplicação do algoritmo de eliminação de estados inacessíveis (Algoritmo 3.10) obtém-se um novo autômato pela exclusão do estado q_5. Pela aplicação do algoritmo de eliminação de estados inúteis (Algoritmo 3.12), obtém-se um novo autômato sem o estado q_6.

Procede-se então à determinação das classes de equivalência partindo-se do conjunto de estados $Q = \{q_0, q_1, q_2, q_3, q_4\}$, pela aplicação do algoritmo 3.17. A partição inicial de Q é:

- Estados finais $= \{q_2, q_4\}$
- Estados não-finais $= \{q_0, q_1, q_3\}$

a) O conjunto $\{q_2, q_4\}$ constitui uma classe de equivalência, pois q_2 e q_4 aceitam as mesmas entradas (nenhuma, neste caso).

b) O conjunto $\{q_0, q_1, q_3\}$ pode ser particionado em $\{q_0\}$ e $\{q_1, q_3\}$, pois seus elementos transitam com entradas diferentes: q_0 transita com $\{a, b\}$ e q_1 e q_3 transitam com $\{c, d, e\}$.

c) $\{q_1, q_3\}$ constitui uma classe de equivalência, pois ambos os estados transitam com as mesmas entradas (no caso, $\{c, d, e\}$) e, além disso:

- $\delta(q_1, c) = q_3$ e $\delta(q_3, c) = q_1$;
- $\delta(q_1, d) = q_1$ e $\delta(q_3, d) = q_3$;
- $\delta(q_1, e) = q_2$ e $\delta(q_1, e) = q_4$, sendo que, conforme (a), q_2 e q_4 são equivalentes.

O autômato resultante deste exemplo é idêntico ao obtido no Exemplo 3.38 (Figura 3.45) □

O método funciona pelo refinamento sucessivo dos conjuntos de estados indistinguíveis para valores crescentes do comprimento das cadeias analisadas. No entanto, a aplicação desse método é um pouco trabalhosa, uma vez que exige a verificação cuidadosa da equivalência para cadeias de comprimento k entre quaisquer dois estados. Note-se, por exemplo, que o simples fato de um estado A aceitar como entrada um subconjunto das entradas aceitas por um estado B não necessariamente implica a distinguibilidade entre ambos, uma vez que as entradas não aceitas em A podem eventualmente não conduzir a estados finais.

Tendo em vista este importante resultado, pode-se retornar ao objetivo inicial desta seção e apresentar o teorema e o algoritmo que tratam da minimização do número de estados de um autômato finito.

Teorema 3.13 (Autômato mínimo) *Seja $M = (Q, \Sigma, \delta, q_0, F)$ um autômato finito. Então, existe um autômato finito $M' = (Q', \Sigma, \delta', q_0, F')$ que possui o menor número possível de estados, tal que $L(M) = L(M')$. Além disso, M' é único.*

Justificativa A obtenção do autômato finito mínimo M' equivalente a M pode ser feita com base no Algoritmo 3.18, que determina as classes de equivalência de M.

As classes de equivalência são denotadas através do nome de um estado de M (qualquer, já que todos são equivalentes), circundado por colchetes, como em $[q]$. A demonstração de que M' é mínimo, e também único, pode ser encontrada em [46].

Algoritmo 3.18 (Minimização de estados) *Minimização do número de estados de um autômato finito.*

- Entrada: um autômato finito $M = (Q, \Sigma, \delta, q_0, F)$;
- Saída: um autômato finito $M' = (Q', \Sigma, \delta', q_0', F')$, tal que $L(M') = L(M)$ e M' não contenha qualquer par de estados equivalentes;
- Método:

1. Construir as partições produzidas pelas relações de equivalência $\overset{0}{\equiv}$, $\overset{1}{\equiv}$, ..., $\overset{k}{\equiv}$, $\overset{k+1}{\equiv}$, até que, para algum valor de k, as partições produzidas pelas relações $\overset{k}{\equiv}$ e $\overset{k+1}{\equiv}$ sejam idênticas.
2. Construir M' tal que:

- Q': o conjunto das classes de equivalência $[q]$ determinadas no passo (1);
- δ': $\{([q], \sigma) = [r] \mid \delta(q, \sigma) = r\}$;
- q_0': $[q_0]$, ou seja, o estado correspondente à classe de equivalência que contém $q_0 \in Q$;
- F': $\{[q] \mid q \in F\}$, ou seja, os estados correspondentes às classes de equivalência que contêm algum $q \in F$.

■

Exemplo 3.38 Considere o autômato finito da Figura 3.44:

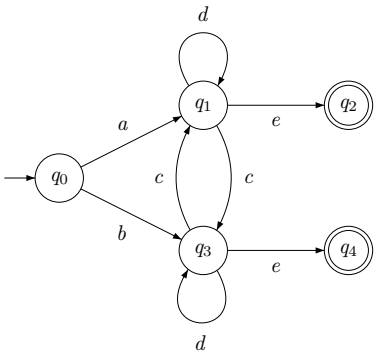

Figura 3.44: Autômato finito original para o Exemplo 3.38

As sucessivas partições do conjunto de estados Q, produzidas pelas relações $\overset{k}{\equiv}, k = 0, 1, 2$, são as seguintes:

$$\begin{aligned} \overset{0}{\equiv} &: \{q_0, q_1, q_3\}, \{q_2, q_4\} \\ \overset{1}{\equiv} &: \{q_0\}, \{q_1, q_3\}, \{q_2, q_4\} \\ \overset{2}{\equiv} &: \{q_0\}, \{q_1, q_3\}, \{q_2, q_4\} \end{aligned}$$

Logo, $Q' = \{[q_0], [q_1], [q_2]\}$ e o autômato resultante é apresentado na Figura 3.45:

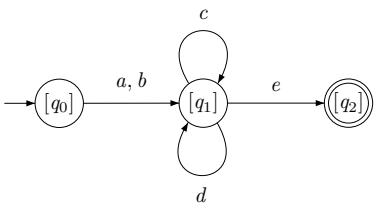

Figura 3.45: Autômato finito mínimo equivalente ao da Figura 3.44

□

Com o intuito de tornar mais simples a identificação das classes de equivalência em autômatos finitos, apresenta-se a seguir um método alternativo, porém equipotente ao método original.

Este outro método opera em dois passos. No primeiro, eliminam-se do autômato os não-determinismos, os estados inacessíveis e os estados inúteis. No segundo, criam-se classes de equivalência com base no critério da coincidência do conjunto de entradas aceitas pelos possíveis pares de estados considerados.

O método descrito pode ser enunciado na forma do Algoritmo 3.19, uma alternativa simples e prática para a minimização de autômatos finitos. Como o algoritmo é baseado na análise exaustiva de todos os possíveis pares de estados de um autômato M, torna-se conveniente representar tais pares na forma de uma matriz, considerada apenas da diagonal principal (inclusive) para cima, uma vez que, para efeito de análise da equivalência de estados, o par (q_i, q_j) e o par (q_j, q_i), com $i \neq j$, são idênticos.

Seja, portanto, M um autômato finito com n estados. A Tabela 3.43 mostra uma forma de representar todos os possíveis pares de estados de M, sem repetição de pares e sem repetição de estados dentro de um mesmo par.

	q_1	q_2	...	q_{n-1}	q_n
q_0	(q_0, q_1)	(q_0, q_2)	...	(q_0, q_{n-1})	(q_0, q_n)
q_1		(q_1, q_2)	...	(q_1, q_{n-1})	(q_1, q_n)
...				...	...
q_{n-2}				(q_{n-2}, q_{n-1})	(q_{n-2}, q_n)
q_{n-1}					(q_{n-1}, q_n)

Tabela 3.43: Representação dos pares de estados de um autômato M com n estados

Adicionalmente, a notação $(q_i, q_j) \xrightarrow{\sigma} (q_m, q_n)$ é usada para indicar que as duas seguintes condições são simultaneamente verificadas por M:

1. $\delta(q_i, \sigma) = q_m$, e
2. $\delta(q_j, \sigma) = q_n$.

Algoritmo 3.19 (Minimização de autômatos) *Método prático para a minimização do número de estados de um autômato finito.*

- Entrada: Um autômato finito determinístico M, com função de transição total, cujos pares de estados estão representados conforme a Tabela 3.43.
- Saída: Uma partição do conjunto de estados Q de M, correspondente às maiores classes de equivalência encontradas em M;
- Método:

    1. Marcar, na tabela, todos os pares do tipo $(q_a, q_b), q_a \in F, q_b \in (Q - F)$ como não-equivalentes ("$\not\equiv$");
    2. Para cada um dos pares de estados restantes (q_a, q_b) (escolhidos arbitrariamente), fazer:

– Se para toda entrada σ aceita por q_a e q_b:

* Se $\delta(q_a, \sigma) = \delta(q_b, \sigma)$, ou

* Se $\delta(q_a, \sigma) \neq \delta(q_b, \sigma)$, mas $\delta(q_a, \sigma)$ e $\delta(q_b, \sigma)$ forem equivalentes.

Então marcar o par (q_a, q_b), na tabela, como equivalente ("$\equiv$"); caso contrário, marcar o par como não-equivalente ("$\not\equiv$"); em seguida, deve-se verificar se existem pares cuja relação de equivalência esteja na dependência do resultado obtido e, em caso afirmativo, marcar os respectivos pares na tabela de forma correspondente;

Caso não seja possível concluir pela equivalência (ou não) de um par de estados, prosseguir com a análise de outros pares, deixando o par corrente na dependência dos resultados que forem obtidos para os demais pares;

3. Marcar os pares restantes, se houver, como equivalentes ("$\equiv$");

4. A inspeção dos pares marcados indica as classes de equivalência obtidas.

Observe-se, neste algoritmo, que a exigência de que a função de transição δ seja total dispensa a verificação de que os estados de cada par considerado aceitem exatamente as mesmas entradas.

Observe-se também que, se houver um ciclo de pares para cujos membros a decisão dependa sempre da de outro par, esses pares devem todos pertencer à mesma classe de equivalência, pois não são distinguíveis em nenhuma instância.

Finalmente, cumpre notar que uma forma conveniente de se registrar e de se manipular a dependência de um par em relação a outros, cuja relação de equivalência ainda não foi estabelecida, e também de se concluir, posteriormente, pela equivalência de um par a partir da verificação da equivalência de um outro par, é através do uso de listas ligadas, em que o primeiro elemento da lista representa o par em relação ao qual os demais apresentam dependências. Essa estrutura de dados mostra-se particularmente conveniente na realização, em forma de programa de computador, do referido algoritmo.

Exemplo 3.39 Considere-se o autômato finito determinístico com a função de transição total da Tabela 3.44.

	δ	a	b
$\rightarrow$	q_0	q_1	q_6
	q_1	q_2	q_3
(f)	q_2	q_2	q_3
	q_3	q_4	q_2
(f)	q_4	q_2	q_3
(f)	q_5	q_4	q_5
	q_6	q_4	q_4

Tabela 3.44: Autômato original do Exemplo 3.39

A Tabela 3.45 representa todos os possíveis pares de estados desse autômato, e também indica a partição inicial de seu conjunto de estados (finais x não-finais).

	q_1	q_2	q_3	q_4	q_5	q_6
q_0		$\not\equiv$		$\not\equiv$	$\not\equiv$	
q_1	-	$\not\equiv$		$\not\equiv$	$\not\equiv$	
q_2	-	-	$\not\equiv$			$\not\equiv$
q_3	-	-	-	$\not\equiv$	$\not\equiv$	
q_4	-	-	-	-		$\not\equiv$
q_5	-	-	-	-	-	$\not\equiv$

Tabela 3.45: Partição inicial dos estados do autômato da Tabela 3.44

Passa-se, então, a considerar cada um dos pares não marcados dessa tabela (escolhidos arbitrariamente). [4]

- $(q_0, q_1) \xrightarrow{a} (q_1, q_2) \not\equiv$
  Como q_1 e q_2 não são equivalentes (ver Tabela 3.45), marca-se o par (q_0, q_1) como "$\not\equiv$" e torna-se desnecessária a análise das transições desses estados com a entrada b.

- $(q_0, q_3) \xrightarrow{a} (q_1, q_4) \not\equiv$
  Similar ao item acima. O par (q_0, q_3) é marcado como "$\not\equiv$".

- $(q_1, q_3) \xrightarrow{a} (q_2, q_4)$?
  $(q_1, q_3) \xrightarrow{b} (q_3, q_2) \not\equiv$
  Apesar de ainda não se dispor de nenhuma informação sobre o par (q_2, q_4), o par (q_3, q_2) já foi determinado como sendo não-equivalente (ver tabela 3.45). Logo, marca-se o par (q_1, q_3) como "$\not\equiv$".

- $(q_0, q_6) \xrightarrow{a} (q_1, q_4) \not\equiv$
  Como q_1 e q_4 não são equivalentes (ver tabela 3.45), marca-se o par (q_0, q_6) como "$\not\equiv$" e torna-se desnecessária a análise das transições desses estados com a entrada b.

- $(q_1, q_6) \xrightarrow{a} (q_2, q_4)$?
  $(q_1, q_6) \xrightarrow{b} (q_3, q_4) \not\equiv$
  Apesar de ainda não se dispor de nenhuma informação sobre o par (q_2, q_4), o par (q_3, q_4) já foi determinado como sendo não-equivalente (ver tabela 3.45). Logo, marca-se o par (q_1, q_6) como "$\not\equiv$".

[4] Com base nas informações já processadas sobre cada par de estados, o símbolo "$\equiv$" indica equivalência, "$\not\equiv$" indica não-equivalência, e "?" indica indefinição e dependência.

- $(q_3, q_6) \xrightarrow{a} (q_4, q_4) \equiv$
  $(q_3, q_6) \xrightarrow{b} (q_2, q_4)$?
  Neste caso, q_3 e q_6 transitam para o mesmo estado q_4 com a entrada a. Por outro lado, ainda não se dispõe de nenhuma informação sobre o par (q_2, q_4). Assim, a equivalência do par (q_3, q_6) fica condicionada à verificação da equivalência do par (q_2, q_4). O par (q_3, q_6) não recebe nenhuma marcação neste momento.

- $(q_2, q_4) \xrightarrow{a} (q_2, q_2) \equiv$
  $(q_2, q_4) \xrightarrow{b} (q_3, q_3) \equiv$
  Os estados q_2 e q_4 transitam com as mesmas entradas para estados idênticos (com a entrada a para q_2 e com a entrada b para q_3). Logo, esses estados são equivalentes e o par recebe a marcação "$\equiv$" na tabela. Além disso, conclui-se que o par (q_3, q_6) (ver item acima) é equivalente, e o mesmo deve ser marcado como "$\equiv$".

- $(q_2, q_5) \xrightarrow{a} (q_2, q_4) \equiv$
  $(q_2, q_5) \xrightarrow{b} (q_3, q_5) \not\equiv$
  Apesar de o par (q_2, q_4) ser equivalente (ver os dois itens anteriores), o par (q_3, q_5) já foi determinado como sendo não-equivalente (ver Tabela 3.45). Logo, marca-se o par (q_2, q_5) como "$\not\equiv$".

- $(q_4, q_5) \xrightarrow{a} (q_2, q_4) \equiv$
  $(q_4, q_5) \xrightarrow{b} (q_3, q_5) \not\equiv$
  Similar ao item acima. O par (q_4, q_5) é marcado como "$\not\equiv$".

Ao término do algoritmo, a Tabela 3.46 resume o resultado da análise.

	q_1	q_2	q_3	q_4	q_5	q_6
q_0	$\not\equiv$	$\not\equiv$	$\not\equiv$	$\not\equiv$	$\not\equiv$	$\not\equiv$
q_1	-	$\not\equiv$	$\not\equiv$	$\not\equiv$	$\not\equiv$	$\not\equiv$
q_2	-	-	$\not\equiv$	$\equiv$	$\not\equiv$	$\not\equiv$
q_3	-	-	-	$\not\equiv$	$\not\equiv$	$\equiv$
q_4	-	-	-	-	$\not\equiv$	$\not\equiv$
q_5	-	-	-	-	-	$\not\equiv$

Tabela 3.46: Resultado final da análise da equivalência de estados para o autômato da Tabela 3.44

As classes de equivalência desse autômato são, portanto, $\{q_0\}, \{q_1\}, \{q_2, q_4\}, \{q_3, q_6\}$ e $\{q_5\}$. O autômato resultante (ver Tabela 3.47) possui cinco estados, denominados respectivamente $[q_0]$, $[q_1]$, $[q_2, q_4]$, $[q_3, q_6]$ e $[q_5]$, e corresponde à versão mínima do autômato da Tabela 3.44.

	δ'	a	b
$\rightarrow$	$[q_0]$	$[q_1]$	$[q_3, q_6]$
	$[q_1]$	$[q_2, q_4]$	$[q_3, q_6]$
(f)	$[q_2, q_4]$	$[q_2, q_4]$	$[q_3, q_6]$
	$[q_3, q_6]$	$[q_2, q_4]$	$[q_2, q_4]$
(f)	$[q_5]$	$[q_2, q_4]$	$[q_5]$

Tabela 3.47: Autômato mínimo equivalente ao da Tabela 3.44

□

Note-se, no Algoritmo 3.19, a exigência de que o autômato a ser minimizado possua função de transição δ total. Tal exigência visa unicamente a garantir que todos os pares de estados, quaisquer que sejam os estados considerados, possam sempre ser comparados em relação a todas as entradas.

No entanto, ao tornar total a função de transição de algum autômato cuja função de transição seja parcial, isso normalmente implica a incorporação de um estado inútil ao autômato, o qual acaba sendo agrupado com outros estados inúteis eventualmente existentes no autômato original e preservado na versão mínima correspondente. Caso o autômato original possua estados inacessíveis, estes também serão agrupados em um único estado no autômato mínimo equivalente.

Assim, o autômato mínimo obtido é tal que a função de transição é total e, além disso, é isento de estados equivalentes. Eventualmente, a eliminação de estados inúteis, e também de estados inacessíveis, em um autômato minimizado de acordo com o Algoritmo 3.19, poderá resultar em um número ainda menor de estados, porém em um autômato cuja função de transição seja parcial. Isso não significa que o autômato inicialmente obtido não seja o mínimo, mas apenas que ele é o autômato mínimo com função de transição total e sem estados equivalentes.

Dois estados A e B de um autômato finito são ditos equivalentes se, em última instância, o conjunto de cadeias aceitas em cada um deles for o mesmo. Considere-se a **linguagem aceita a partir de um estado** X como sendo definida da seguinte forma:

$$L(X) = \{w \in \Sigma^* \mid (X, w) \vdash^* (\epsilon, q_F), q_F \in F\}$$

Logo, é fácil perceber que $A \equiv B$ se e somente se $L(A) = L(B)$. Esse resultado pode simplificar a verificação da equivalência de estados para os quais a determinação da linguagem aceita em cada um deles seja uma tarefa simples de ser feita (por inspeção visual ou pela aplicação de um método qualquer).

Exemplo 3.40 Considere-se o autômato da Figura 3.46:

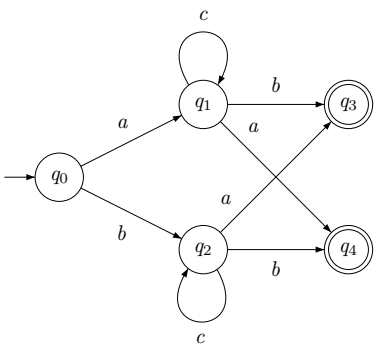

Figura 3.46: Autômato original do Exemplo 3.40

Uma rápida inspeção visual permite concluir que:

- $L(q_0) = (a \mid b)c^*(a \mid b)$
- $L(q_1) = c^*(a \mid b)$
- $L(q_2) = c^*(a \mid b)$

- $L(q_3) = \epsilon$
- $L(q_4) = \epsilon$

Portanto, como $L(q_1) = L(q_2)$ e $L(q_3) = L(q_4)$, então $q_1 \equiv q_2$ e $q_3 \equiv q_4$, e a versão mínima corresponde à apresentada na Figura 3.47.

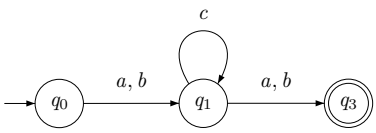

Figura 3.47: Autômato mínimo equivalente ao da Figura 3.46

□

O Teorema 3.13 apresenta dois resultados fundamentais para as linguagens regulares, conforme abaixo. Dada uma linguagem regular L qualquer, então:

1. Existe um autômato finito mínimo que aceita L. Em outras palavras, não existe nenhum outro autômato, com um número inferior de estados, que aceite L;

2. O autômato finito mínimo que aceita L é único. Isso significa que não existem dois autômatos finitos com o mesmo número de estados, porém com funções de transição distintas, que aceitam a linguagem L.

A existência e unicidade de um autômato finito mínimo para toda e qualquer linguagem regular L permite, entre outros resultados, estabelecer critérios para determinar se um conjunto de linguagens regulares representa a mesma linguagem: basta obter as versões mínimas dos autômatos finitos que reconhecem cada uma dessas linguagens, e verificar se são todos iguais. Em caso afirmativo, as linguagens são todas idênticas. Caso contrário, não são todas idênticas.

3.7 Transdutores Finitos

Estudadas as definições, notações e conceitos fundamentais associados aos autômatos finitos, analisam-se a seguir algumas importantes extensões dessa classe de autômatos.

Inicialmente, considerem-se os denominados **transdutores finitos**. Os autômatos finitos, como foi visto, são simples dispositivos de reconhecimento de sentenças e, como tal, são muito pobres quanto às suas informações de saída: a única por eles emitida é a da aceitação ou rejeição da cadeia analisada. Com o intuito de ampliar a aplicabilidade dos autômatos finitos, foram definidas extensões que associam, a cada sentença de entrada, uma correspondente cadeia de saída sobre um segundo alfabeto, eventualmente distinto do alfabeto de entrada.

A associação de símbolos de um alfabeto de saída com a movimentação do autômato finito no reconhecimento de uma sentença pode ser feita de duas formas distintas: a partir da seqüência de estados percorridos ou das transições de que se compõe o autômato finito no qual se baseia o transdutor em questão. O primeiro caso caracteriza as denominadas Máquinas de Moore, e o segundo as chamadas Máquinas de Mealy.

Formalmente, um transdutor finito do tipo **Máquina de Moore** é definido como sendo uma sétupla $T_{Moore} = (Q, \Sigma, \Delta, \delta, \lambda, q_0, F)$ sobre um autômato finito $M = (Q, \Sigma, \delta, q_0, F)$, em que Δ é o **alfabeto de saída** do transdutor e $\lambda : Q \rightarrow \Delta$ é a **função de transdução**[5] de T.

Na notação dos Diagramas de Estado, cada estado do autômato finito é rotulado com a identificação do símbolo do alfabeto de saída que deve ser gerado toda vez que o estado for atingido.

Exemplo 3.41 Seja T um transdutor finito do tipo Máquina de Moore:

$$\begin{aligned} T &= (Q, \Sigma, \Delta, \delta, \lambda, q_0, F) \\ Q &= \{q_0, q_1\} \\ \Sigma &= \{a, b, c\} \\ \Delta &= \{1\} \\ \delta &= \{(q_0, a) \rightarrow q_1, (q_1, b) \rightarrow q_1, (q_1, c) \rightarrow q_0\} \\ \lambda &= \{q_0 \rightarrow 1, q_1 \rightarrow \epsilon\} \\ F &= \{q_1\} \end{aligned}$$

O grafo correspondente a esse transdutor é apresentado na Figura 3.48:

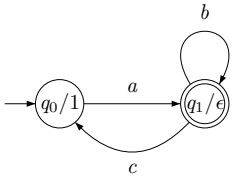

Figura 3.48: Transdutor do tipo Máquina de Moore

A linguagem aceita por esse transdutor é $ab^*(cab^*)^*$, ou seja, uma seqüência de uma ou mais cadeias ab^* separadas pelo símbolo c. A função de transição λ, neste caso, faz com que o transdutor emita o símbolo "1" toda vez que estiver iniciando o reconhecimento de uma nova cadeia com o formato ab^*. Assim, T funciona como um contador do número de subcadeias ab^* presentes na cadeia de entrada. Como exemplo, a Tabela 3.48 apresenta um conjunto de cadeias que são respectivamente aceitas e geradas por T.

Sentença aceita	Cadeia Gerada
abbcabbbcab	111
abbbcab	11
acacaca	1111
a	1

Tabela 3.48: Sentenças aceitas e cadeias geradas pelo transdutor do tipo Máquina de Moore T

□

Um transdutor finito do tipo **Máquina de Mealy**, por sua vez, é definido como sendo uma sétupla $T_{Mealy} = (Q, \Sigma, \Delta, \delta, \lambda, q_0, F)$ sobre um autômato finito $M = (Q, \Sigma, \delta, q_0, F)$, em que Δ é o **alfabeto de saída** do transdutor e $\lambda : Q \times \Sigma \rightarrow \Delta$ é a **função de transdução**[6] de T.

[5] Alternativamente, alguns autores preferem adotar $\lambda : Q \rightarrow \Delta^*$.
[6] Alguns autores adotam $\lambda : Q \times \Sigma \rightarrow \Delta^*$.

No caso das Máquinas de Mealy, associam-se os símbolos do alfabeto de saída às transições, e não aos estados, como ocorre com as Máquinas de Moore (o domínio da função λ se altera para $Q \times \Sigma$).

Exemplo 3.42 Seja T um transdutor finito do tipo Máquina de Mealy:

$$\begin{aligned} T &= (Q, \Sigma, \Delta, \delta, \lambda, q_0, F) \\ Q &= \{q_0, q_1\} \\ \Sigma &= \{a, b, c\} \\ \Delta &= \{a, b, c\} \\ \delta &= \{(q_0, a) \rightarrow q_1, (q_1, b) \rightarrow q_1, (q_1, c) \rightarrow q_0\} \\ \lambda &= \{(q_0, a) \rightarrow ab, (q_1, b) \rightarrow \epsilon, (q_1, c) \rightarrow c\} \\ F &= \{q_1\} \end{aligned}$$

O grafo correspondente é apresentado na Figura 3.49:

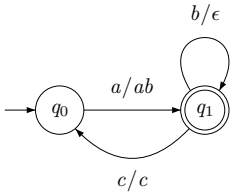

Figura 3.49: Transdutor do tipo Máquina de Mealy

Como se pode notar, o autômato finito que serve de base para esse transdutor é o mesmo do exemplo anterior. Assim, a linguagem aceita por ambos os transdutores é a mesma. No entanto, o transdutor deste exemplo mapeia as subcadeias ab^* aceitas pelo autômato finito em cadeias do tipo ab, mantendo o símbolo c como separador na sentença de saída. Observe que a função de transdução λ foi definida, neste caso, sobre Δ^* (devido à utilização da cadeia ϵ). A Tabela 3.49 apresenta alguns exemplos de cadeias respectivamente aceitas e geradas por T:

Sentença aceita	Cadeia Gerada
abbcabbbcab	*abcabcab*
abbbcab	*abcab*
acacaca	*abcabcabcab*
a	*ab*

Tabela 3.49: Sentenças aceitas e cadeias geradas pelo transdutor do tipo Máquina de Mealy T

□

Apesar de se tratar de dois modelos distintos de transdutores finitos, pode-se demonstrar a plena equivalência de ambos: toda e qualquer Máquina de Moore pode ser simulada por uma Máquina de Mealy e vice-versa. Dessa maneira, portanto, a opção por um ou outro tipo de máquina pode ser feita levando-se em conta exclusivamente a conveniência de manipulação e a facilidade de representação obtidas conforme o caso em questão.

Teorema 3.14 (Equivalência dos transdutores) *Toda Máquina de Mealy pode ser simulada por uma Máquina de Moore, e vice-versa.*

Justificativa Pode ser encontrada em [46]. ■

Exemplo 3.43 Considere-se a linguagem $L_1 = xx^*(-xx^*)^*$, definida sobre o alfabeto $\{x, -\}$. Considere-se também a linguagem L_2, definida sobre o alfabeto de saída $\{x, y, \#\}$, de tal forma que as cadeias de L_2 reproduzem na saída as cadeias de L_1, com as seguintes modificações:

- As subcadeias de entrada xx^* que contiverem três ou menos símbolos x devem ser reproduzidas de forma idêntica na saída (com um, dois ou três símbolos x);
- As subcadeias de entrada xx^* que contiverem quatro ou mais símbolos x devem ser reproduzidas na saída como $xxxy$;
- Todos os símbolos "-" da cadeia entrada devem ser substituídos pelo símbolo "#" na cadeia de saída.

A Tabela 3.50 apresenta exemplos de cadeias de entrada e correspondentes cadeias de saída:

Sentença aceita	Cadeia Gerada
$x - x$	$x\#x$
$xxx - xxxx$	$xxx\#xxxy$
$xxxxxx - xxx - xx$	$xxxy\#xxx\#xx$
$x - xx - xxx - xxxx - xxxxx$	$x\#xx\#xxx\#xxxy\#xxxy$

Tabela 3.50: Sentenças aceitas e cadeias geradas pelos transdutores do Exemplo 3.43

Os transdutores finitos das Figuras 3.50 e 3.51 — respectivamente Máquina de Mealy e Máquina de Moore — são equivalentes, pois possuem autômatos subjacentes que reconhecem a mesma linguagem L_1 (apesar de serem diferentes) e geram a mesma linguagem L_2, conforme as especificações acima.

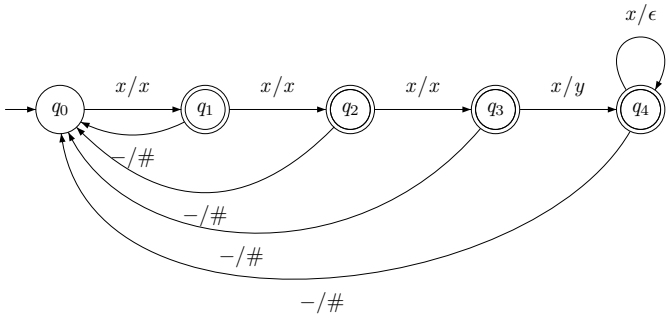

Figura 3.50: Transdutor do tipo Máquina de Mealy do Exemplo 3.43

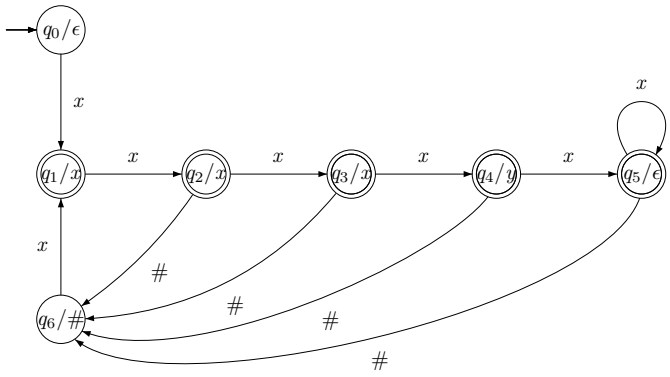

Figura 3.51: Transdutor do tipo Máquina de Moore equivalente ao transdutor do tipo Máquina de Mealy da Figura 3.50

□

Além de definirem a linguagem de entrada associada ao autômato finito subjacente, os transdutores, sejam eles de um tipo ou de outro, definem uma segunda linguagem, denominada **linguagem de saída**, denotada por $L(T)$, correspondente ao conjunto das sentenças sobre Δ que são geradas quando do reconhecimento de sentenças pertencentes a $L(M)$, onde M é o autômato finito em que o transdutor é baseado. Demonstra-se que a classe de linguagens que pode ser gerada por um transdutor finito corresponde exatamente à classe de linguagens que pode ser reconhecida pelo autômato finito em que ele se baseia: a classe das linguagens regulares.

3.8 Linguagens que não são Regulares

No Capítulo 2 foi mencionado que as linguagens regulares, ou do tipo 3, constituiriam um subconjunto próprio das linguagens livres de contexto, ou do tipo 2. Para verificar a veracidade dessa proposição, é necessário antes demonstrar a existência de linguagens que não são regulares.

Uma das formas mais usuais para se provar que determinadas linguagens não são regulares é através da utilização do "Pumping Lemma", a seguir apresentado. Em linhas gerais, o "Pumping Lemma"[7] estabelece uma propriedade que é sempre verdadeira para toda e qualquer linguagem regular. Caso a linguagem considerada não exiba tal propriedade, pode-se concluir imediatamente que a mesma não é regular.

Teorema 3.15 ("Pumping Lemma" para linguagens regulares) *Seja L um conjunto regular infinito. Então existe uma constante n, dependente apenas de L, tal que, para quaisquer sentenças $w \in L$, com $|w| \geq n$, w pode ser subdividida em três subcadeias x, y e z, de tal forma que $w = xyz, 1 \leq |y|, |xy| \leq n$, ou seja, $1 \leq |y| \leq n$, e, além disso, $xy^i z \in L, \forall\, i \geq 0$.*

Justificativa O reconhecimento de qualquer cadeia $x \in L$, com $|x| \geq n$, sendo L aceita

[7] Referido por alguns autores em português como "Lema do Bombeamento".

por um autômato finito M com n estados, ocorre percorrendo-se pelo menos dois estados idênticos entre as $n+1$ configurações assumidas por M durante o reconhecimento dos primeiros n símbolos de x.

Seja $w = a_1 a_2 ... a_m, \left|w\right| = m, m \geq n$. A seqüência abaixo ilustra a evolução da configuração do autômato M no reconhecimento de w:

$$q_0 \overset{a_1}{\rightarrow} q_1 \overset{a_2}{\rightarrow} q_2 \overset{a_3}{\rightarrow} q_3 ... \overset{a_{n-1}}{\rightarrow} q_{n-1} \overset{a_n}{\rightarrow} q_n \overset{a_{n+1}}{\rightarrow} q_{n+1} ... \overset{a_{m-1}}{\rightarrow} q_{m-1} \overset{a_m}{\rightarrow} q_m$$

onde $q_0 ... q_m$ são os estados sucessivamente percorridos por M (não necessariamente distintos entre si).

Considerando-se os $n+1$ estados inicialmente percorridos por M $(q_0, q_1 ... q_n)$, é fato que pelo menos dois desses estados devem ser idênticos. Existem então duas possibilidades extremas a serem consideradas, no que diz respeito à localização desses estados idênticos na seqüência:

1. A distância entre eles é a menor possível: $(q_i, a_k ... a_m) \vdash (q_j, a_{k+1} ... a_m), q_i = q_j, j \leq n$;

2. A distância entre eles é a maior possível: $(q_0, a_1 ... a_m) \vdash (q_n, a_{n+1} ... a_m), q_0 = q_n$.

Reescrevendo-se w como xyz, em que x corresponde à parte da cadeia de entrada que leva M à primeira ocorrência de um estado repetido na seqüência, e y corresponde à parte da cadeia que leva M à sua segunda ocorrência, tem-se que:

- $\left|y\right| \geq 1$;
- $\left|xy\right| \leq n$;
- Portanto, $1 \leq \left|y\right| \leq n$, pois $\left|y\right| \leq \left|xy\right|$.

Caso haja, entre os $n+1$ estados inicialmente percorridos pelo autômato, três ou mais ocorrências de um mesmo estado, permanecem válidos os limites superior (n) e inferior (1) previamente determinados para o comprimento da cadeia y.

Como se pode perceber, o fato de a subcadeia y levar o autômato de um estado q_i, anterior ao seu reconhecimento, para o mesmo estado $q_j = q_i$, posterior ao seu reconhecimento, caracteriza como um ciclo o caminho percorrido pelos estados de M, com os símbolos de y. Pelo fato de se tratar de um ciclo, repetições arbitrárias do mesmo conduzem ao reconhecimento de sentenças também pertencentes à linguagem definida pelo autômato. Dessa forma, todas as sentenças do tipo $xy^i z$, com $i \geq 0$, pertencem necessariamente a $L(M)$.

O "Pumping Lemma" se baseia no fato de que cadeias com uma certa quantidade mínima de símbolos não podem ser reconhecidas por autômatos finitos sem que haja repetição de configurações. Por esse motivo, a "limitação" da quantidade de estados é explorada para demonstrar a existência de ciclos, e estes, por sua vez, para demonstrar que outras sentenças devem necessariamente pertencer à linguagem.

O "Pumping Lemma" das linguagens regulares estabelece a propriedade de que, dada uma sentença de comprimento mínimo n pertencente a esta linguagem, é sempre possível identificar, na subcadeia formada pelos seus n primeiros símbolos, uma nova subcadeia cujo comprimento está entre 1 e n, de tal modo que repetições arbitrárias da mesma geram sentenças que também pertencem à linguagem definida.

Assim, a constante n corresponde ao número de estados do autômato finito utilizado para definir a linguagem regular. No entanto, como é sabido, uma mesma linguagem regular pode ser definida por autômatos finitos distintos, os quais podem possuir, eventualmente, um número de estados também distintos.

Suponha-se L uma linguagem regular e infinita que seja aceita por dois autômatos finitos distintos M_1 e M_2, $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ e $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$. Sejam $n_1 = \left|Q_1\right|$ e $n_2 = \left|Q_2\right|$, e suponha-se $n_1 > n_2$. Se $w \in L$ e $\left|w\right| \geq n_1$, então é claro que $\left|w\right| \geq n_2$ e w pode ser escolhida para verificar o "Pumping Lemma" em M_2. Por outro lado, se $\left|w\right| \geq n_2$, como L é infinita, então é fato que existe uma outra cadeia $z \in L$, $\left|z\right| \geq n_1$, que pode ser usada para verificar o "Pumping Lemma" em M_1. Logo, é indiferente a escolha de M_1 ou M_2 e, conseqüentemente, de n_1 ou n_2, para demonstrar a validade do "Pumping Lemma" para a linguagem L.

Por outro lado, é natural que se questione a existência de um valor para a constante n que independa do autômato analisado, e que possa, portanto, ser considerado como inerente à linguagem. Considerando-se a existência de um autômato finito mínimo que reconhece uma dada linguagem regular L (ver Seção 3.6), é natural que se considere o número de estados do correspondente autômato finito como o valor n inerente à linguagem L.

Observe-se que, embora o teorema prove a existência da constante n, a sua aplicação em casos práticos não exige que se determine o valor dessa constante. ■

Exemplo 3.44 Considere-se um autômato finito M com cinco estados distintos, e suponha-se que M efetue a análise de uma cadeia $p \in L(M), \left|p\right| = 5$. Claramente, M deverá percorrer seis estados durante o reconhecimento da cadeia. Não obstante, como M apresenta apenas cinco estados distintos, é evidente que pelo menos dois (eventualmente mais) dos estados assumidos por M durante o reconhecimento de p são idênticos. Considere-se agora uma cadeia $q \in L(M), \left|q\right| = 20$. Da mesma forma, analisando-se os seis primeiros estados percorridos por M, constata-se que obrigatoriamente haverá pelo menos dois estados repetidos entre eles, correspondentes ao reconhecimento dos cinco primeiros símbolos de q. □

Exemplo 3.45 Seja $M = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, F)$ um autômato finito sem transições em vazio. Se $ab \in L(M)$, então a seqüência de configurações assumidas por M durante a análise dessa cadeia deve, necessariamente, corresponder a alguma das seguintes possibilidades:

1. $(q_0, ab) \vdash (q_0, b) \vdash (q_0, \epsilon)$
2. $(q_0, ab) \vdash (q_0, b) \vdash (q_1, \epsilon)$
3. $(q_0, ab) \vdash (q_1, b) \vdash (q_0, \epsilon)$
4. $(q_0, ab) \vdash (q_1, b) \vdash (q_1, \epsilon)$

Considerando-se os demais elementos de M desconhecidos, pode-se apenas especular sobre a real seqüência que corresponde à aceitação da cadeia ab por M. De qualquer forma, as seguintes conclusões são válidas:

1. Se $(q_0, ab) \vdash (q_0, b) \vdash (q_0, \epsilon)$, então pelo menos uma das seguintes possibilidades é verdadeira:
   a) $x = \epsilon, y = ab, z = \epsilon$;
   b) $x = a, y = b, z = \epsilon$;
   c) $x = \epsilon, y = a, z = b$.
2. Se $(q_0, ab) \vdash (q_0, b) \vdash (q_1, \epsilon)$, então:

a) $x = \epsilon, y = a, z = b.$

3. $(q_0, ab) \vdash (q_1, b) \vdash (q_0, \epsilon)$

   a) $x = \epsilon, y = ab, z = \epsilon.$

4. $(q_0, ab) \vdash (q_1, b) \vdash (q_1, \epsilon)$

   a) $x = a, y = b, z = \epsilon.$

Portanto, qualquer que seja o caso, é sempre possível identificar, na cadeia ab, cujo comprimento coincide com o número de estados do autômato que a aceita, uma subcadeia y, de comprimento maior ou igual a 1 e menor ou igual a 2, que provoca um ciclo na seqüência de movimentações executada pelo autômato. □

Exemplo 3.46 Considere-se o autômato da Figura 3.52:

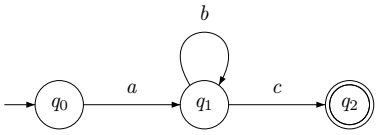

Figura 3.52: Aplicação do "Pumping Lemma" ao autômato finito que aceita ab^*c

A aplicação das propriedades enunciadas através do "Pumping Lemma" a este autômato podem ser verificadas através do uso de cadeias de comprimento maior ou igual a 3, uma vez que ele possui três estados:

- Considere-se a cadeia $w = abc, \left|w\right| = 3$. Então, w pode ser reescrito como $xyz, \left|xy\right| \leq 3, 1 \leq \left|y\right| \leq 3$ e, finalmente, $xy^iz \in L, \forall\, i \geq 0$. Nesse caso, deve-se escolher $x = a, y = b, z = c$. Assim, $xz = ac, xyyz = abbc, xyyyz = abbbc$ etc. são todas cadeias que pertencem a L.

- Considere-se a cadeia $w = abbbc, \left|w\right| = 5$. Então, w pode ser reescrito como $xyz, \left|xy\right| \leq 3, 1 \leq \left|y\right| \leq 3$ e, finalmente, $xy^iz \in L, \forall\, i \geq 0$. Nesse caso podem-se fazer três escolhas distintas de subdivisão da cadeia w, todas em conformidade com os critérios do "Pumping Lemma":

  – $x = a, y = b, z = bbc$. As cadeias $(a)(b)^*(bbc)$ estão contidas em L.

  – $x = a, y = bb, z = bc$. As cadeias $(a)(bb)^*(bc)$ estão contidas em L.

  – $x = ab, y = b, z = c$. As cadeias $(ab)(b)^*(c)$ estão contidas em L.

  Nem todas as subdivisões de uma cadeia w geram cadeias que produzem cadeias que pertencem à linguagem. Note-se, em particular, no exemplo acima, que seria possível relacionar, entre as subdivisões possíveis da cadeia de comprimento 5, as seguintes alternativas:

  i) $x = \epsilon, y = a, z = bbbc$;

  ii) $x = \epsilon, y = ab, z = bbc$;

  iii) $x = \epsilon, y = abb, z = bc.$

Em todos esses casos, xy^iz gera cadeias que não pertencem a L. Qualquer que seja a cadeia escolhida, o "Pumping Lemma" garante apenas que, se ela possuir comprimento mínimo, então ao menos uma subdivisão xyz da mesma será possível de ser feita, de modo que todas as cadeias xy^iz também pertençam à linguagem. □

A principal aplicação do "Pumping Lemma" consiste na demonstração da existência de linguagens não-regulares. Outras aplicações importantes podem ser encontradas na demonstração de certas questões decidíveis da classe das linguagens regulares.

A demonstração de que uma dada linguagem não é regular pode ser feita por contradição, da seguinte forma:

1. Admite-se inicialmente, por hipótese, que a linguagem sob análise seja regular;

2. Através de manipulações, demonstra-se que a linguagem não exibe as propriedades descritas pelo "Pumping Lemma";

3. Conclui-se, por contradição, que a hipótese não é verdadeira, e portanto que a linguagem não é regular.

Na prática, a aplicação do método se inicia com a escolha de uma sentença da linguagem que possua o comprimento mínimo exigido pelo "Pumping Lemma". Como normalmente o valor de n não é conhecido, admite-se n como parâmetro do problema, e expressa-se a sentença escolhida em função de n. Na prática, a determinação do valor numérico de n é desnecessária, bastando o conhecimento teórico da sua existência. A seguir, através de manipulações da sentença selecionada, busca-se identificar outras cadeias dela derivadas conforme as premissas do "Pumping Lemma", mas que violem a definição da linguagem. Havendo sucesso nesse passo, conclui-se que o "Pumping Lemma" das linguagens regulares não é válido para a linguagem considerada e, portanto, que tal linguagem não é regular.

O "Pumping Lemma" não se aplica aos casos em que se deseja demonstrar que uma determinada linguagem é regular. Para que isso fosse uma possibilidade concreta, seria necessário mostrar que, para toda e qualquer sentença da linguagem (xyz), toda e qualquer outra sentença obtida a partir dela, através da aplicação do "Pumping Lemma" (xy^iz), também pertence à mesma linguagem. Na prática isso é inviável, uma vez que:

- As linguagens que exibem maior interesse são infinitas (linguagens finitas são trivialmente regulares); [8]

- A quantidade de cadeias geradas a partir de cada senteça considerada é infinita.

A demonstração de que uma linguagem é regular só pode ser feita através da apresentação de uma enumeração, de uma expressão regular, de um autômato finito ou de uma gramática regular que defina a linguagem.

Exemplo 3.47 Seja $L = \{a^k b^k \mid k \geq 0\}$. Supondo que L seja uma linguagem regular, tome-se a sentença $a^n b^n$, onde n é a constante definida pelo "Pumping Lemma". Essa sentença pertence a L e possui comprimento $2n$, portanto maior ou igual a n. De acordo com o "Lemma", essa sentença pode ser decomposta em três subcadeias x, y e z, tais que $xyz = a^n b^n, \left|xy\right| \leq n, \left|y\right| \geq 1$.

[8] Linguagens trivialmente regulares são reconhecidas por autômatos finitos isentos de ciclos, e portanto a eles o "Pumping Lemma" não se aplica, já que exprime uma propriedade periódica da linguagem, e em linguagens finitas não se podem detectar periodicidades. Em outras palavras, nas cadeias xyz, $y = \epsilon$, e isso contraria a hipótese do "Pumping Lemma"

Logo, $y = a^i, 1 \leq i \leq n$, e xyz pode ser reescrito como $a^{n-i}a^ib^n$. No entanto, nenhuma das seguintes cadeias pertence a L:

1. $xy^0z = a^{n-i}b^n$
2. $xyyz = a^{n-i}a^ia^ib^n = a^{n+i}b^n$

uma vez que as ocorrências do símbolo a estão desbalanceadas em relação às ocorrências dos símbolos b. Logo, L não é regular. □

Exemplo 3.48 Seja $L = \{0^k10^k \mid k \geq 1\}$ e considere-se uma sentença w de comprimento suficientemente longo pertencente a esta linguagem, $w = 0...010...0$. Admitindo-se que seja possível escrever w como xyz, tem-se que $1 \leq y \leq n$, onde n é a constante de L, e y pode assumir uma das cinco formas seguintes:

1. $y = 1$
2. $y \in 0^+$
3. $y \in 0^+1$
4. $y \in 10^+$
5. $y \in 0^+10^+$

Como é fácil perceber, se $y = 1$, então $xy^0z \notin L$, pois faltará o símbolo "1", obrigatório em todas as sentenças de L.

Se $y \in 0^+$, então $xyyz \notin L$, pois haverá quantidades diferentes do símbolo "0" antes e após o símbolo "1" na sentença.

Se $y \in 0^+1, y \in 10^+$ ou, ainda, $y \in 0^+10^+$, então $xyyz \notin L$, uma vez que $xyyz$ terá mais que um único símbolo "1". Fica assim demonstrado, por contradição, que L não é uma linguagem regular, visto que não atende ao "Pumping Lemma". □

Exemplo 3.49 Considere-se a linguagem $L = \{a^{k*k} \mid k \in \mathbb{Z}_+\}$. De acordo com essa definição, as sentenças de L são seqüências formadas por símbolos a de comprimento 1, 4, 9, 16 etc. Seja n a constante de L e considere-se a sentença a^{n*n}.

Essa cadeia pode ser reescrita como xyz, em que $1 \leq \left|y\right| \leq n$. Pelo "Pumping Lemma", se $xyz \in L$, então $xyyz \in L$. Considerando a sentença $xyyz$, tem-se que $n^2 < \left|xyyz\right| \leq n^2 + n$. Por outro lado, $n^2 + n < (n+1)^2$, portanto, $n^2 < \left|xyyz\right| < (n+1)^2$. Ora, isso contraria a hipótese de que o comprimento de todas as sentenças dessa linguagem correspondem ao quadrado de algum número inteiro positivo, uma vez que não existe $i \in \mathbb{Z}_+$ tal que $n^2 < i^2 < (n+1)^2, \forall\, n \in \mathbb{Z}_+$. Fica assim demonstrado, por contradição, que L não é uma linguagem regular. □

Exemplo 3.50 Seja $L = \{a^kb^kc^k \mid k \geq 1\}$. Supondo que L seja uma linguagem regular, tome-se a sentença $a^nb^nc^n$, onde n é a constante definida pelo "Pumping Lemma". Claramente essa sentença pertence a L. Mas, de acordo com o "Lemma", essa sentença pode ser decomposta em três subcadeias x, y e z, tais que $xyz = a^nb^nc^n, \left|xy\right| \leq n, \left|y\right| \geq 1$.

Logo, $y = a^i, 1 \leq i \leq n$, e xyz pode ser reescrito como $a^{n-i}a^ib^nc^n$. No entanto, nenhuma das seguintes cadeias pertence a L:

1. $xy^0z = a^{n-i}b^nc^n$
2. $xyyz = a^{n-i}a^ia^ib^nc^n = a^{n+i}b^nc^n$

uma vez que as ocorrências do símbolo a estão desbalanceadas em relação às dos símbolos b e c. Logo, L não é regular. Observe-se a semelhança da presente demonstração com a que foi efetuada para a linguagem a^kb^k no Exemplo 3.47. □

Exemplo 3.51 Considere-se $L = \{a^k \mid k \geq 1$ é um número primo$\}$. Admitindo-se que L seja uma linguagem regular, tome-se a sentença a^m, onde m é o primeiro número primo superior à constante n definida pelo "Pumping Lemma". Logo, $m > n$. De acordo com o "Lemma", como $\left|a^m\right| = m \geq n$, essa sentença pode ser decomposta em três subcadeias x, y e z, com $xyz = a^m, \left|xy\right| \leq n, \left|y\right| \geq 1$.

Além disso, $xy^i z \in L, \forall\, i \geq 0$. Em particular, pode-se fazer $i = m+1$. Logo, de acordo com o "Lemma", a cadeia $xy^{m+1}z$ deveria pertencer a L. No entanto, $\left|xy^{m+1}z\right| = \left|xyzy^m\right| = \left|xyz\right| + \left|y^m\right|$. Como $\left|xyz\right| = m$ e $\left|y^m\right| = m * \left|y\right|$, então $\left|xy^{m+1}z\right| = m + m * \left|y\right| = m * (1 + \left|y\right|)$.

Esse resultado mostra que o comprimento de $xy^{m+1}z$, ou seja, $m*(1+\left|y\right|)$, não é um número primo, uma vez que:

- Ele é divisível por m, pois $\dfrac{m * (1 + \left|y\right|)}{m} = (1 + \left|y\right|)$;
- $m \neq 1$, pois $n \geq 1$ e $m > n$;
- $m \neq m * (1 + \left|y\right|)$, pois, de acordo com o "Pumping Lemma", $\left|y\right| \geq 1$.

Logo, L não é regular. □

3.9 Propriedades de Fechamento

Conforme definido na Seção 1.2, diz-se que uma determinada classe de linguagens é fechada em relação a uma operação se da aplicação da operação a quaisquer linguagens dessa classe resultar sempre uma linguagem que também pertença à classe em questão.

O estudo de uma classe de linguagens do ponto de vista das operações em relação às quais ela é fechada é muito importante, uma vez que auxilia, na prática, na determinação da classe de linguagens a que uma certa linguagem possa ou não pertencer.

A seguir são apresentadas, na forma de teoremas, as principais propriedades de fechamento para as linguagens regulares.

Teorema 3.16 (Fecho na união, concatenação e fecho) *A classe das linguagens regulares é fechada em relação às operações de união, concatenação e fechamento reflexivo e transitivo.*

Justificativa Imediata, a partir da definição dos conjuntos regulares. ■

Teorema 3.17 (Fecho na complementação) *A classe das linguagens regulares é fechada em relação à operação de complementação.*

Justificativa Seja $L(M)$ a linguagem aceita por um autômato finito determinístico M=(Q, Δ, δ, q_0, F), sendo δ uma função total, e considere-se $\Delta \subseteq \Sigma$. Como se pode perceber pela Figura 3.53, $\Sigma^* - L = (\Sigma^* - \Delta^*) \cup (\Delta^* - L(M))$.

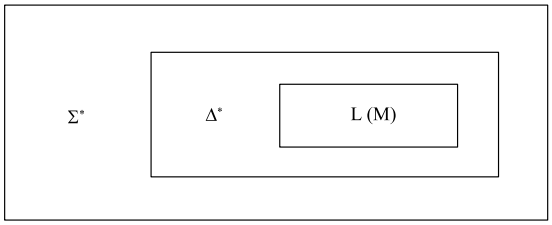

Figura 3.53: Representação de Σ^*, Δ^* e $L(M)$

$\Sigma^* - \Delta^*$ contém as sentenças que possuem pelo menos um elemento de $\Sigma - \Delta$, e $\Delta^* - L(M)$ as sentenças sobre Δ rejeitadas por M.

A linguagem $\Delta^* - L(M)$ é aceita pelo autômato $M' = (Q, \Delta, \delta, q_0, Q - F)$, em que os estados finais de M tornam-se não-finais em M' e vice-versa. Assim, se $x \in L(M)$, ou seja, se $\delta(q_0, x) \in F$, então $x \notin L(M')$, uma vez que $\delta(q_0, x) \notin (Q - F)$.

Logo, conclui-se que, se L for uma linguagem regular, então $\Delta^* - L(M)$ será também uma linguagem regular, uma vez que ela é aceita pelo autômato finito M'.

Por outro lado, a linguagem $\Sigma^* - \Delta^*$, de acordo com a sua interpretação (conjunto de "sentenças que possuem pelo menos um elemento de $\Sigma - \Delta$"), pode ser reescrita como:

$$\Sigma^* - \Delta^* = \Sigma^*(\Sigma - \Delta)\Sigma^*$$

Portanto, como decorrência do fechamento das linguagens regulares sobre as operações de fechamento reflexivo e transitivo e de concatenação, é possível afirmar que $\Sigma^* - \Delta^*$ é regular.

Finalmente, $(\Sigma^* - \Delta^*) \cup (\Delta^* - L(M))$ é também uma linguagem regular, uma vez que $(\Sigma^* - \Delta^*)$ e $(\Delta^* - L(M))$ são fechadas em relação à operação de união. ■

Exemplo 3.52 Considere-se M o autômato finito determinístico da Figura 3.54, $L(M) = (ab\big|c)d^*e^*$.

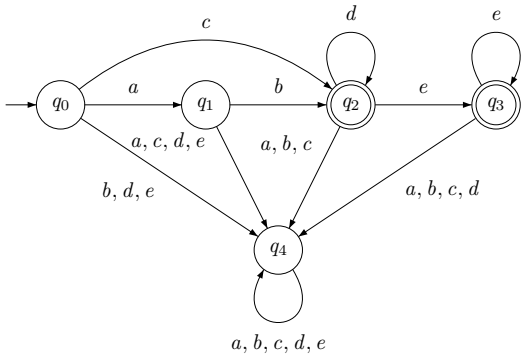

Figura 3.54: Autômato finito que aceita L

Através do método apresentado na demonstração do Teorema 3.17 obtém-se M', representado na Figura 3.55, de modo que $L(M') = \overline{L(M)}$, com $\Sigma = \Delta = \{a, b, c, d, e\}$.

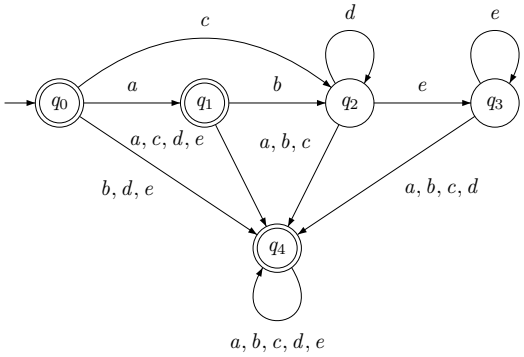

Figura 3.55: Autômato finito que aceita $\overline{L}$ (ver Figura 3.54)

□

Observe-se que foi fundamental, para a demonstração deste teorema, a garantia de que a linguagem L seja sempre definida através de um autômato finito determinístico e com função de transição δ total. Assim, qualquer que seja a cadeia de entrada, ela será sempre completamente consumida por M e, em conseqüência, a aceitação ou rejeição da mesma por M dependerá apenas do tipo do último estado assumido por M (final ou não-final). Ao trocar estados finais por não-finais e vice-versa em M', todas as cadeias de entrada sempre serão também completamente consumidas por M' e, por fim, as cadeias aceitas por M (ou seja, as que o conduzem a um estado final) serão rejeitadas por M' (por o conduzirem a um estado não-final) e vice-versa.

Caso essa providência não tivesse sido tomada, certas cadeias de entrada poderiam não ser completamente esgotadas, e, nessa situação, a simples inversão de estados seria insuficiente para garantir que M' aceitasse o complemento da linguagem aceita por M. Suponha-se, por exemplo, que M rejeite uma cadeia em um estado final sem esgotá-la. Neste caso, M' iria também rejeitá-la, uma vez que o estado não-final de M teria se convertido em final de M', porém a cadeia de entrada não teria sido esgotada. O correto, no entanto, seria que M' aceitasse essa cadeia, como se pode notar através do Exemplo 3.53.

Exemplo 3.53 O autômato finito não-determinístico da Figura 3.56 aceita a linguagem ab^*.

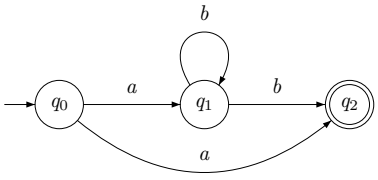

Figura 3.56: Autômato finito M que aceita $L = ab^*$

Caso o método fosse aplicado a este autômato, na forma em que ele se encontra, o resultado corresponderia ao apresentado na Figura 3.57.

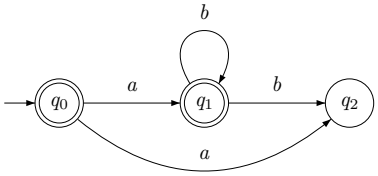

Figura 3.57: Autômato finito M' que aceita $L' = ab^*$

Como se pode perceber, $L'(M') = ab^* = L(M)$ e, portanto, frustra-se o objetivo de se obter um autômato que aceite o complemento de L. Tal resultado decorre da não observância da exigência de se eliminar, *a priori*, os não-determinismos de M, além do fato de que δ não é uma função total□

Teorema 3.18 (Fecho na intersecção) *A classe das linguagens regulares é fechada em relação à operação de intersecção.*

Justificativa Considere-se a linguagem L_1 sobre Σ_1, e L_2 sobre Σ_2, sendo $\Sigma_1, \Sigma_2 \subseteq \Sigma$. Então, considerando-se as complementações em relação a Σ, a seguinte relação é verdadeira (Lei de De Morgan, ver Teorema 1.1):

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

Portanto, a regularidade da linguagem resultante da intersecção de duas outras linguagens regulares depende da preservação da regularidade pelas operações de união e complemento. Como esse fato já foi constatado nos Teoremas 3.16 e 3.17, é possível afirmar, com base no presente teorema, que $L_1 \cap L_2$ será necessariamente uma linguagem regular. ■

Teorema 3.19 (Fecho na substituição) *A classe das linguagens regulares é fechada em relação à operação de substituição.*

Justificativa Considere-se R uma expressão regular que denota o conjunto regular L sobre Σ, e seja $s : \Sigma \rightarrow 2^{\Delta^*}$ uma função de substituição. Para cada $\sigma \in \Sigma$, seja $s(\sigma)$ o conjunto regular determinado pela expressão regular R_σ.

Substituindo-se cada ocorrência em R de $\sigma \in \Sigma$ por R_σ, obtém-se uma nova expressão regular R' em que cada símbolo do alfabeto original é substituído pela correspondente expressão regular. É possível constatar que $R' = s(R)$. A prova pode ser feita através da indução sobre o número de operadores presentes em R (ver [46]), notando que:

1. $s(L_1 \cup L_2) = s(L_1) \cup s(L_2)$
2. $s(L_1 L_2) = s(L_1)s(L_2)$
3. $s(L_1^*) = (s(L_1))^*$

■

Exemplo 3.54 Considere-se $L = 0^*(1^* \mid 2^*)$, e a substituição s abaixo definida:

- $s(0) = a \mid b$

- $s(1) = c^* d$
- $s(2) = e$

Então, $s(L) = s(0^*(1^* \mid 2^*)) = s(0^*)s(1^* \mid 2^*) = (s(0))^*(s(1^*) \mid s(2^*)) = (s(0))^*((s(1))^* \mid (s(2))^*) = (a \mid b)^*((c^* d)^* \mid (e)^*)$. □

Teorema 3.20 (Fecho no homomorfismo e homomorfismo inverso) *A classe das linguagens regulares é fechada em relação às operações de homomorfismo e homomorfismo inverso.*

Justificativa O fechamento em relação à operação de homomorfismo é imediato, uma vez que os homomorfismos são casos particulares de substituições.

A prova do fechamento em relação à operação de homomorfismo inverso pode ser baseada na construção de um autômato finito M' que aceita a imagem homomórfica inversa de uma linguagem regular $L = L(M)$ qualquer. A demonstração formal pode ser feita por indução sobre o comprimento das sentenças $x \in h^{-1}(L)$ em seu reconhecimento por M' (ver [46]).

Seja $M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$ um autômato finito determinístico que aceite L, e considere-se o homomorfismo h:

$$h : \Delta \to \Sigma^*$$

Então, o autômato $M' = (Q, \Delta, \delta', q_0, F)$, com:

$$\delta'(q, a) = \delta(q, h(a)), \quad a \in \Delta$$

aceita $h^{-1}(L)$, ou seja, $L(M') = h^{-1}(L(M))$. Observe-se que a função de transição δ' especificada para M' simula a operação do autômato M através de transições efetuadas diretamente para os estados que sucedem o reconhecimento das cadeias $h(a)$ em cada estado considerado no autômato M. ■

Exemplo 3.55 Seja $L(M) = (ab)^*((cd)^* \mid e^*)$, com M definido conforme a Figura 3.58.

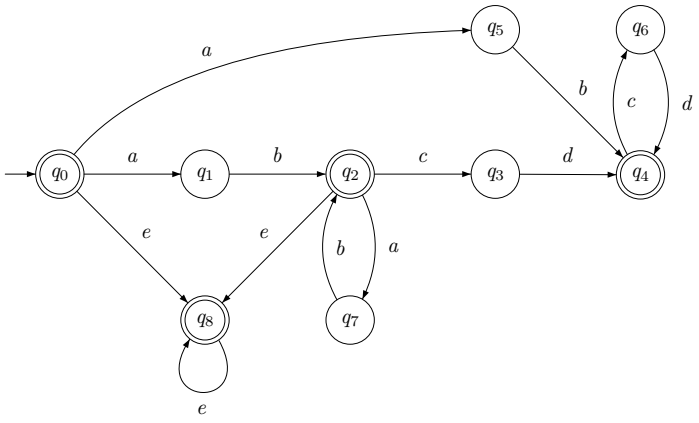

Figura 3.58: Autômato finito que aceita L

Suponha-se que L tenha sido obtida através do homomorfismo h:

- $h(0) = ab$
- $h(1) = cd$
- $h(2) = e$

O autômato M', que aceita a imagem homomórfica inversa de L, é apresentado na Figura 3.59.

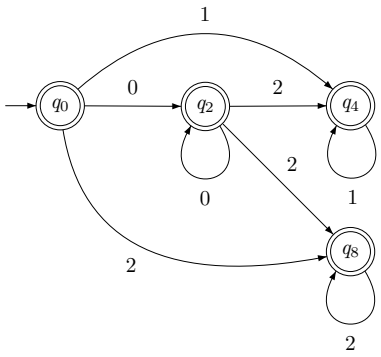

Figura 3.59: Autômato finito que aceita $h^{-1}(L)$

□

Teorema 3.21 (Fecho no quociente) *A classe das linguagens regulares é fechada em relação à operação de quociente com linguagens de tipo arbitrário.*

Justificativa Sejam $M = (Q, \Sigma, \delta, q_0, F)$ um autômato finito que aceita L_1 e L_2 uma linguagem qualquer. Pode-se então definir o autômato $M' = (Q, \Sigma, \delta, q_0, F')$, de modo que $\forall\, q \in Q$:

1. $q \in F'$ se $\exists\, y \in L_2$, tal que $\delta(q, y) \in F$;
2. $q \notin F'$, caso contrário.

Como se pode perceber, $\delta(q_0, x) \in F'$ se e apenas se existir $y \in L_2$, tal que $\delta(\delta(q_0, x), y) \in F$. Logo, $L(M') = L_1/L_2$ é uma linguagem regular. ■

Exemplo 3.56 Sejam as linguagens $L_1 = a^*b$ e $L_2 = a \mid b$. O autômato finito M da Figura 3.60 aceita L_1.

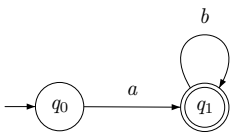

Figura 3.60: Autômato finito que aceita L_1

A linguagem $L = L_1/L_2 = a^*$ é aceita pelo autômato M' da Figura 3.61.

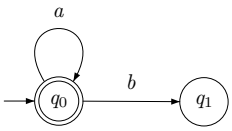

Figura 3.61: Autômato finito que aceita L_1/L_2

De fato,

1. A linguagem aceita por q_1 em M é ϵ. Como $\epsilon \notin L_2$, então $q_1 \notin F'$.
2. A linguagem aceita por q_0 em M é a^*b. Como $a^*b \supset L_2$, então $q_0 \in F'$.

□

Teorema 3.22 (Fecho na reversão) *A classe das linguagens regulares é fechada em relação à operação de reversão de suas sentenças (linguagem reversa).*

Justificativa Seja $M = (Q, \Sigma, \delta, q_0, F)$ um autômato finito que aceita L. O Algoritmo 3.20 mostra como construir $M' = (Q \cup \{q_0'\}, \Sigma, \delta', q_0', F')$, eventualmente não-determinístico, de tal modo que $L(M') = L^R$.

Algoritmo 3.20 (Autômato para $L \Rightarrow$ autômato para L^R) *Construção do autômato finito que aceita L^R a partir do autômato finito que aceita L.*

- Entrada: um autômato finito $M = (Q, \Sigma, \delta, q_0, F)$;
- Saída: um autômato finito $M' = (Q \cup \{q_0'\}, \Sigma, \delta', q_0', F')$, tal que $L(M') = L(M)^R$;
- Método:

    1. Construção de F':

        a) Se $\epsilon \notin L$, então $F' = \{q_0\}$;

        b) Se $\epsilon \in L$, então $F' = \{q_0, q_0'\}$.

    2. Construção de δ':

        c) Se $\delta(q, \sigma) \in F$, então $\delta'(q_0', \sigma) = q$;

        d) Se $\delta(q_a, \sigma) = q_b$, então $\delta'(q_b, \sigma) = q_a$.

Observe que as condições a) e b) são mutuamente exclusivas, e que, no entanto, as condições c) e d) são satisfeitas simultaneamente no caso de transições que conduzem a estados finais. Em outras palavras, se uma transição conduz a um estado final em M, ela dá origem a duas transições em M'.

O algoritmo consiste, na essência, em promover a inversão do sentido das transições de M em M', tomando-se o cuidado de convergir todas as transições que levam a estados

finais de M para um único estado inicial em M'. O estado inicial de M torna-se o estado final de M'. É possível demonstrar formalmente (ver [46]) que:

$$(q_0, x) \vdash^* (q_f, \epsilon) \text{ se e somente se } (q_0', x^R) \vdash^* (q_0, \epsilon).$$

Em outras palavras, $L(M') = L(M)^R = L^R$. ■

Exemplo 3.57 Considere a linguagem a^*bc^* aceita por M, conforme a Figura 3.62:

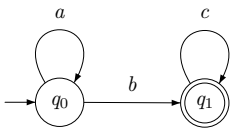

Figura 3.62: Autômato finito que aceita $L = a^*bc^*$

A aplicação do algoritmo descrito conduz à obtenção do autômato M' da figura 3.63:

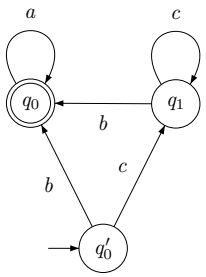

Figura 3.63: Autômato finito que aceita $L^R = (a^*bc^*)^R = c^*ba^*$

Como se pode observar, $L(M') = ba^* \mid cc^*ba^* = c^*ba^* = L(M)^R$. □

Uma das principais aplicações do estudo do fechamento de uma classe de linguagens em relação a um determinado conjunto de operações consiste na possibilidade de se determinar a classe de uma linguagem a partir da decomposição da mesma em linguagens mais simples, de classe conhecida, e que, combinadas por intermédio de operadores que preservam a classe dessas linguagens mais simples, nos permitem inferir diretamente a classe das linguagens resultantes.

Exemplo 3.58 Considere-se a linguagem $L = \{a^*bc^* \mid$ o comprimento das sentenças é maior ou igual a $3\}$. L é regular?

L pode ser representada como $L_1 \cap L_2$, onde $L_1 = \{a^*bc^*\}$ e $L_2 = (a \mid b \mid c)(a \mid b \mid c)(a \mid b \mid c)(a \mid b \mid c)^*$. Como L_1 e L_2 são regulares (pois estão expressas através de expressões regulares), e a classe das linguagens regulares é fechada em relação à operação de intersecção, então L também é regular. De fato, não é difícil perceber que $L = a^*(aab \mid abc \mid bcc)c^*$. □

Exemplo 3.59 Seja a linguagem $L = \{w \in \{a, b, c, d\}^* \mid w$ contém a subcadeia "bb" e w não contém a subcadeia "dd"$\}$. L é regular?

$L = L_1 \cap \overline{L_2}$, onde $L_1 = \{w \in \{a, b, c, d\}^* \mid w$ contém a subcadeia "bb"$\}$ e $L_2 = \{w \in \{a, b, c, d\}^* \mid w$ contém a subcadeia "dd"$\}$:

- $L_1 = (a \mid b \mid c \mid d)^* bb(a \mid b \mid c \mid d)^*$
- $L_1 = (a \mid b \mid c \mid d)^* dd(a \mid b \mid c \mid d)^*$

Como L_1 e L_2 são regulares, e a classe das linguagens regulares é fechada em relação às operações de complemento e intersecção, segue que L também é regular. □

Exemplo 3.60 Considere-se a linguagem L_1 definida sobre o alfabeto $\{a, b\}$, de tal forma que pertencem a L_1 todas as cadeias que podem ser formadas com os símbolos de seu alfabeto, excetuando-se aquelas que contêm exatamente três símbolos a. L_1 é regular?

Não é difícil perceber que $L_1 = \overline{L_2}$, onde $L_2 = b^*ab^*ab^*ab^*$, ou seja, L_1 corresponde à complementação da linguagem que contém todas as cadeias com exatamente três símbolos a (L_2). Portanto, L_1 é regular. Uma expressão regular que representa L_1 é $b^* \mid b^*ab^* \mid b^*ab^*ab^* \mid b^*ab^*ab^*ab^*a(a \mid b)^*$□

Exemplo 3.61 Considere-se o alfabeto $\{a, b, c\}$ e a linguagem L definida de tal forma que suas cadeias satisfazem todas às seguintes regras:

1. Possuem a subcadeia aaa como prefixo;
2. Possuem comprimento total múltiplo de 4;
3. Possuem quantidade par de símbolos c;
4. Não contêm a subcadeia bb.

São exemplos de cadeias pertencentes a L: $aaabcccc$, $aaabcbca$, $aaaa$, $aaaaaaba$, $aaaacccbaaac$ etc. L é regular? Para responder, basta notar que $L = ((L_1 \cap L_2) \cap L_3) \cap \overline{L_4}$, onde:

- L_1 é gerada por $aaa(a \mid b \mid c)^*$:
  Cadeias que possuem aaa como prefixo.
- L_2 é gerada por $((a \mid b \mid c)(a \mid b \mid c)(a \mid b \mid c)(a \mid b \mid c))^*$:
  Cadeias que possuem comprimento total múltiplo de 4.
- L_3 é gerada por $((a \mid b)^* c(a \mid b)^* c(a \mid b)^*)^*$:
  Cadeias que possuem quantidade par de símbolos c.
- L_4 é gerada por $(a \mid b \mid c)^* bb(a \mid b \mid c)^*$:
  Cadeias que contêm a subcadeia bb.

Como L_1, L_2, L_3 e L_4 são regulares, e a classe das linguagens regulares é fechada em relação às operações de intersecção e complementação, conclui-se que L é regular. □

Portanto, a demonstração da regularidade de uma linguagem não precisa ficar restrita à especificação de uma expressão regular, de um autômato finito ou, ainda, de uma gramática regular que definam essa linguagem. A partir da identificação do uso de operações que preservam a regularidade de linguagens mais simples na constituição de linguagens mais complexas, essa demonstração pode ser consideravelmente simplificada.

Foram apresentadas algumas das principais propriedades de fechamento exibidas pelas linguagens regulares. Outras propriedades, com suas respectivas demonstrações, podem ser encontradas em [46], [48] e [53]. Não obstante, outras propriedades estudadas nos capítulos seguintes para as demais classes de linguagens também se aplicam diretamente às linguagens regulares, uma vez que, conforme a Hierarquia de Chomsky, toda linguagem do tipo 3 também é, simultaneamente, uma linguagem do tipo 2, 1 e 0.

3.10 Questões Decidíveis

Para cada uma das classes de linguagens consideradas pela Hierarquia de Chomsky, é possível formular uma série de questionamentos de ordem geral sobre as linguagens a elas pertencentes. Por exemplo, pode-se questionar se uma sentença é aceita por um dado reconhecedor, se uma linguagem é igual a outra, se uma linguagem é infinita, se uma linguagem é não-vazia etc.

Conforme discutido a seguir, vários desses problemas são muito simples de serem resolvidos quando se trata de linguagens regulares. No entanto, quando se consideram outras classes de linguagens, verifica-se que nem sempre algumas dessas questões podem ser respondidas. Por esse motivo, a classe de problemas considerados a seguir é genericamente denominada de **questões decidíveis**.

É importante notar que não costumam ser considerados, no estudo geral das linguagens formais, os casos particulares, ou seja, os problemas particularizados para uma certa linguagem ou grupo de linguagens, como, por exemplo: *"A linguagem L, a seguir apresentada, é vazia?"*.

Ao contrário, o interesse recai sobre problemas gerais parametrizados, que podem ser instanciados para uma ou mais linguagens específicas quando de sua aplicação, como, por exemplo: *"É possível descobrir se uma linguagem regular L, qualquer que seja L, é vazia?"*.

Quando se diz que um problema (ou questão) desse tipo é **decidível**, isso significa que ele sempre tem solução, qualquer que seja a sua instância considerada (ou argumentos aplicados). Mais do que isso, cada questão decidível é caracterizada pela existência de um algoritmo que permite resolver o problema geral com garantias de obtenção do resultado — afirmativo ou negativo, dependendo do caso.

As questões apresentadas neste item referem-se à classe das linguagens regulares. Questões referentes a outras classes de linguagens serão consideradas nos capítulos correspondentes. Cada questão decidível está apresentada na forma de um teorema, acompanhado de um algoritmo que possibilita a automatização da implementação da solução do problema proposto. Certas questões indecidíveis relevantes, próprias de outras classes de linguagens, são apresentadas na forma de teoremas com a respectiva prova formal (ou informal, conforme o caso) de sua indecidibilidade.

Antes de iniciar, convém fazer algumas considerações sobre os critérios de escolha do tipo de representação de linguagens adotado nas demonstrações seguintes. Analogamente ao que foi feito na Seção 3.9, certos resultados estão mostrados com base em um ou outro tipo de representação — expressões regulares, autômatos finitos ou gramáticas lineares à direita —, dependendo de qual seja mais conveniente em cada caso. A escolha de um ou outro tipo de representação não possui implicação sobre os resultados apresentados, uma vez que elas são equivalentes, sendo sempre possível efetuar a conversão de um tipo para outro mecanicamente, se for o caso.

Os três teoremas iniciais referem-se a questões decidíveis cuja demonstração é baseada nos resultados do "Pumpimg Lemma" das linguagens regulares.

Teorema 3.23 (A linguagem é vazia?) *A linguagem L aceita por um autômato finito com n estados é não-vazia se e somente se o autômato aceita pelo menos uma cadeia* $w, |w| < n$.

Justificativa A condição necessária ("aceita uma sentença de comprimento inferior a $n \Rightarrow$ linguagem é não-vazia") é óbvia e não necessita ser demonstrada.

A condição suficiente ("linguagem é não-vazia ⇒ aceita uma sentença de comprimento inferior a n") não é tão óbvia, mas pode ser verificada com auxílio do "Pumping Lemma". Considere-se $w \in L(M), |w| = m$.

Se $m < n$, então nada há para demonstrar, e a hipótese é trivialmente verdadeira.

Se, no entanto, $m \geq n$, então w pode ser reescrita como xyz com $xz \in L(M), y \neq \epsilon, |xz| < m$. Seguem, então, duas possibilidades: $|xz| \geq n$ ou $|xz| < n$. Se $|xz| < n$, a hipótese está demonstrada. Se, por outro lado, $|xz| \geq n$, pode-se agora considerar $w = xz$ e aplicar o "Pumping Lemma" novamente, desta vez sobre tal cadeia.

Através da iteração deste passo, é possível obter cadeias de comprimentos sucessivamente menores, enquanto o comprimento da cadeia anterior for maior ou igual a n. Assim, é possível demonstrar a existência de uma sentença de comprimento inferior a n, pertencente a L. ■

A condição suficiente do Teorema 3.23 pode também ser compreendida através do seguinte raciocínio: partindo-se do estado inicial, se o autômato aceitar pelo menos uma cadeia, então a linguagem é não-vazia. Como o autômato possui n estados, então é necessário que pelo menos um desses estados seja final, e também acessível desde o estado inicial.

Se o estado inicial for simultaneamente final, então a cadeia vazia é aceita e a linguagem aceita pelo autômato é não-vazia. Observe-se ainda que $|\epsilon| < n$, qualquer que seja o valor de n, uma vez que $n \geq 1$.

Se o estado inicial não for simultaneamente final, então será necessário atingir pelo menos um dos outros $n-1$ estados do autômato, o qual deve também ser final. Para isso, bastam cadeias de comprimento máximo $n-1$, inclusive, já que cadeias de comprimento maior ou igual a n possuem ciclos (conforme o "Pumping Lemma"), e não modificam o conjunto de estados que são acessíveis a partir do estado considerado. Logo, se nenhuma cadeia de comprimento menor que n for aceita pelo autômato, isso significa que:

- Não existem estados finais no autômato, ou
- Os estados finais do autômato não são acessíveis desde o estado inicial

e, portanto, a linguagem por ele aceita é vazia.

Em outras palavras, qualquer estado acessível de um autômato finito com n estados é alcançável por meio de cadeias de comprimento máximo $n-1$. Se algum desses estados for final, então a linguagem aceita é não-vazia. Caso contrário, é vazia.

A importância deste teorema se deve ao algoritmo alternativo por ele implicitamente proposto: para determinar se uma linguagem, aceita por um autômato finito com n estados, é não-vazia, basta verificar se o autômato aceita alguma sentença de comprimento entre 0 (inclusive) e $n-1$ (inclusive). Como a quantidade de cadeias sobre um alfabeto finito que satisfazem a tal condição é finita (conforme mostrado a seguir), então pode-se demonstrar que o problema proposto pode sempre ser resolvido através da pesquisa exaustiva da aceitação dessas cadeias por M: se pelo menos uma dessas cadeias for aceita, constata-se que a linguagem é não-vazia, não sendo necessário testar outras cadeias de comprimento maior. Se nenhuma dessas cadeias for aceita, pode-se concluir que a linguagem é vazia, sem testar quaisquer outras cadeias.

Para um autômato finito com n estados, cujo alfabeto de entrada tenha m símbolos, a quantidade de cadeias que devem ser testadas é dada pela fórmula:

$$\sum_{i=0}^{n-1} m^i$$

pois, conforme pode ser verificado na Tabela 3.51, essa fórmula representa a quantidade total de cadeias distintas cujos comprimentos estão entre 0 (inclusive) e $n-1$ (inclusive), e que podem ser construídas a partir de um alfabeto com m símbolos.

Comprimento	Cadeias distintas	Cadeias distintas
0	1	m^0
1	m	m^1
2	$m * m$	m^2
3	$m * m * m$	m^3
...	...	...
$n-1$	$m * m * m * ... * m$	m^{n-1}

Tabela 3.51: Quantidade de cadeias que podem ser obtidas a partir de um alfabeto com m símbolos, com comprimento entre 0 e $n-1$

Exemplo 3.62 Seja L uma linguagem regular sobre o alfabeto $\{a, b, c\}$ e aceita por um autômato finito M com três estados. Então, para determinar se L é não-vazia, basta verificar se alguma das seguintes cadeias é aceita por M:

- Comprimento 0 (uma cadeia): ϵ
- Comprimento 1 (três cadeias): a, b, c
- Comprimento 2 (nove cadeias): $aa, ab, ac, ba, bb, bc, ca, cb, cc$

Se alguma dessas 13 (= 1 + 3 + 9) cadeias for aceita por M, então L será não-vazia. Caso contrário, será vazia. □

Teorema 3.24 (A linguagem é infinita?) *A linguagem L aceita por um autômato finito com n estados é infinita se e somente se o autômato aceitar pelo menos uma cadeia $w \in \Sigma^*, n \leq |w| < 2n$.*

Justificativa A condição "se" (aceita pelo menos uma cadeia $w, n \leq |w| < 2n \Rightarrow$ linguagem infinita) pode ser facilmente deduzida através do "Pumping Lemma": como $|w| \geq n$, então w pode ser reescrita como xyz, e $xy^iz \in L, \forall\, i \geq 0$. Logo, L é infinita.

A condição "somente se" (linguagem infinita $\Rightarrow$ aceita pelo menos uma cadeia $w, n \leq |w| < 2n$) é demonstrada, por contradição, a seguir.

Se L é infinita, então com certeza existem cadeias de comprimento maior ou igual a n (pois a quantidade de cadeias com comprimento menor ou igual a n é finita). Considere-se $w \in L(M), |w| \geq n$.

Se $|w| < 2n$, então não há nada a demonstrar e a hipótese é trivialmente verdadeira.

Se $|w| \geq 2n$, então, de acordo com o "Pumping Lemma", $w = xyz, |xy| \leq n, 1 \leq |y| \leq n$. Logo, a cadeia xz também pertence a $L, |xz| < |w|, |xz| \geq n$ (pois, como $|w| \geq 2n$ e, na pior das hipóteses, $|y| = n$, então $|xz| = |w| - |y| \geq n$).

Duas possibilidades podem então ocorrer com a cadeia xz: ou $|xz| \geq 2n$ ou $|xz| < 2n$.

Se $|xz| < 2n$, então a hipótese é verdadeira e o teorema está demonstrado.

Se $|xz| \geq 2n$, pode-se considerar agora $w = xz$ e aplicar o "Pumping Lemma" novamente sobre essa cadeia. Através da iteração deste passo, enquanto o comprimento de w for maior ou igual a $2n$, é possível obter cadeias de comprimentos sucessivamente menores, porém sempre de comprimento maior ou igual a n. Logo, necessariamente existe uma cadeia pertencente à linguagem, de comprimento maior ou igual a n e menor que $2n$, e o teorema está demonstrado.

Assim, se a linguagem for infinita, ela deverá obrigatoriamente conter pelo menos uma cadeia de comprimento entre n (inclusive) e $2n$ (exclusive). ■

A condição "somente se" do Teorema 3.24 pode também ser compreendida da seguinte forma: por se tratar de uma linguagem infinita, e portanto não-vazia, o autômato correspondente aceita pelo menos uma cadeia w_0, $0 \leq |w_0| < n$ (ver Teorema 3.23).

Por outro lado, como se trata de uma linguagem infinita, então é fato que este autômato possui pelo menos um ciclo, correspondente à cadeia y, $1 \leq |y| \leq n$ (conforme o "Pumping Lemma").

Logo, a combinação desses resultados (ou seja, o "bombeamento" da cadeia y na cadeia w_0, resultando em uma nova cadeia cujo comprimento corresponde à soma dos comprimentos mínimos e máximos das outras duas) garante a existência de pelo menos uma cadeia w_1, $1 < |w_1| < 2n - 1$, que é aceita pelo autômato.

Se $|w_1| \geq n$, então a condição está provada. Se $|w_1| < n$, pode-se "bombear" y novamente, desta vez em w_1, resultando na cadeia w_2, $2 < |w_2| < 2n - 1$.

A iteração desse passo, enquanto $w_i < n$, garante a existência de uma cadeia w_j, aceita pelo autômato, tal que $n \leq |w_j| < 2n - 1$, como se quer demonstrar.

Em outras palavras, a existência de ciclos acessíveis desde o estado inicial garante que o autômato aceita pelo menos uma cadeia w tal que $n \leq |w| < 2n - 1$.

A principal aplicação deste teorema se encerra no algoritmo por ele sugerido, o qual permite determinar se a linguagem aceita por um autômato finito com n estados é infinita ou não: basta verificar se o autômato aceita alguma cadeia de comprimento entre n (inclusive) e $2n - 1$ (inclusive). Como a quantidade de cadeias com essa característica é finita, conclui-se ser sempre possível determinar se uma linguagem regular é infinita ou não, bastando para isso analisar, exaustivamente, se alguma dessas cadeias pertence à linguagem definida.

A quantidade de cadeias que devem ser testadas em um autômato com n estados e cujo alfabeto de entrada possui m símbolos é dada pela fórmula:

$$\sum_{i=n}^{2n-1} m^i$$

Exemplo 3.63 Seja L uma linguagem regular sobre o alfabeto $\{a, b\}$ e aceita por um autômato finito M com 2 estados. Então, para saber se L é não-vazia, basta verificar se alguma das seguintes cadeias é aceita por M:

- Comprimento 2 (quatro cadeias): aa, ab, ba, bb
- Comprimento 3 (oito cadeias): aaa, aab, aba, abb, baa, bab, bba, bbb

Se alguma dessas 12 (= 4 + 8) cadeias for aceita por M, então L é infinita. Caso contrário, conclui-se que a linguagem é finita. □

Teorema 3.25 (A linguagem é finita?) *A linguagem L aceita por um autômato finito com n estados é finita se e somente se o autômato não aceita nenhuma sentença w tal que $n \leq |w| < 2n$.*

Justificativa Decorre diretamente do teorema anterior. L é infinita se e somente se o autômato finito correspondente aceita pelo menos uma cadeia $w, n \leq |w| < 2n$. Logo, se não existir nenhuma cadeia que satisfaça a essa condição, a linguagem L será finita. Para determinar se, além de finita, L é não-vazia, basta verificar se o autômato finito correspondente aceita pelo menos uma cadeia de comprimento menor do que n (Teorema 3.23). ■

A Tabela 3.52 resume os resultados até aqui obtidos.

L: uma linguagem aceita por um autômato finito com n estados		
$\exists\, w \in L, \|w\| < n$?	$\exists\, w \in L, n \leq \|w\| < 2n$?	L é ...
Sim	Sim	**Infinita**
Sim	Não	**Finita, não-vazia**
Não	Não	**Finita, vazia**
Não	Sim	N.A. (contradição)

Tabela 3.52: Cardinalidade de uma linguagem regular

Exemplo 3.64 Seja uma linguagem L_1 sobre o alfabeto $\{a\}$, aceita por um autômato M_1 com três estados. Para determinar se L_1 é vazia, basta verificar se alguma das cadeias pertencentes ao seguinte conjunto é aceita por M_1: $X = \{\epsilon, a, aa\}$. Para determinar se L_1 é infinita, deve-se verificar as cadeias do conjunto $Y = \{aaa, aaaa, aaaaa\}$.

Seja o autômato M_1, representado na Figura 3.64.

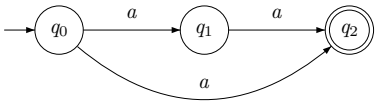

Figura 3.64: Autômato M_1 que aceita $L_1 = \{a, aa\}$, finita e não-vazia

É fácil perceber, neste caso, que $L_1(M_1) = \{a, aa\}$ é finita e não-vazia. De fato, as cadeias a, aa de X são aceitas por M_1. No entanto, nenhuma das cadeias $aaa, aaaa, aaaaa$ de Y são aceitas por M_1.

Suponha-se agora M_2, correspondente ao autômato da Figura 3.65:

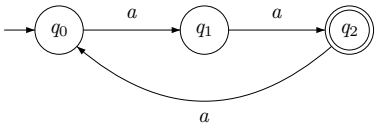

Figura 3.65: Autômato M_2 que aceita $L_2 = aa(aaa)^*$, infinita

A linguagem $L_2(M_2)$ é, neste caso, $aa(aaa)^*$, e portanto infinita. A infinitude de L_2 é comprovada pelo fato de M_2 aceitar a cadeia $aaaaa$ de Y. O fato de M_2 aceitar aa de X indica que L_2 é não-vazia.

Por último, considere-se M_3 como sendo o autômato da Figura 3.66:

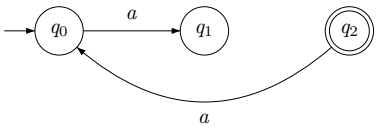

Figura 3.66: Autômato M_3 que aceita $L = \varnothing$, finita e vazia

Nenhuma das cadeias ϵ, a, aa de X é aceita por M_3. Logo, como se pode comprovar observando-se a Figura 3.66, L_3 é vazia (e portanto finita). □

Teorema 3.26 (A cadeia pertence?) *Seja L uma linguagem regular sobre $\Delta, \Delta \subseteq \Sigma$, e $\alpha \in \Sigma^*$ uma cadeia. Então, a questão $\alpha \in L$ é decidível.*

Justificativa Seja $M = (Q, \Delta, \delta, q_0, F)$ tal que $L = L(M)$. O Algoritmo 3.21 mostra como decidir se a cadeia α pertence ou não à linguagem L.

Algoritmo 3.21 (A cadeia pertence?) *Determina se uma cadeia é sentença da linguagem definida por um autômato finito.*

- Entrada: um autômato finito $M = (Q, \Delta, \delta, q_0, F), \Delta \subseteq \Sigma,$ e uma cadeia $\alpha \in \Sigma^*$;
- Saída: Se $\alpha \in L(M)$, SIM; caso contrário, NÃO;
- Método:

    1. Obter $M' = (Q, \Delta, \delta', q_0, F')$, isento de transições em vazio, tal que $L(M') = L(M)$;
    2. Determinar $\delta'(q_0, \alpha)$. Se $\delta'(q_0, \alpha) \in F'$, então SIM; caso contrário, NÃO.

■

O Algoritmo 3.21 garante que qualquer cadeia pode ser analisada em um número finito de passos (ou tempo finito de processamento) em um autômato finito. Para isso, é suficiente garantir que o mesmo seja isento de transições em vazio, o que implica a inexistência de ciclos formados exclusivamente por transições desse tipo, as quais poderiam, eventualmente, provocar um processamento interminável da cadeia de entrada.

Exemplo 3.65 Considere-se o autômato da Figura 3.67, que possui um ciclo formado por transições em vazio.

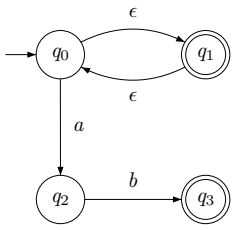

Figura 3.67: Autômato com ciclo de transições em vazio

Os movimentos executados por esse autômato na análise da cadeia ba não permitem que o mesmo pare em qualquer configuração, final ou não-final, como mostra a seguinte seqüência:

$$(q_0, ba) \vdash (q_1, ba) \vdash (q_0, ba) \vdash (q_1, ba) \vdash (q_0, ba) \vdash (q_1, ba) \vdash ...$$

O autômato equivalente, da Figura 3.68, é isento de transições em vazio, e, portanto, de ciclos formados por transições em vazio.

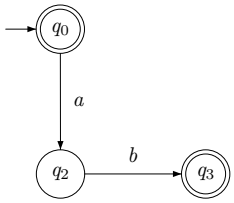

Figura 3.68: Autômato equivalente ao da Figura 3.67, porém isento de ciclos formados por transições em vazio

Esse autômato atinge a seguinte configuração de parada, para a mesma cadeia ba de entrada, após executar zero movimentações:

$$(q_0, ba)$$

Portanto, por não se tratar de uma configuração final, a cadeia ba é rejeitada e não pertence à linguagem definida pelos autômatos das Figuras 3.67 e 3.68. □

Teorema 3.27 (As linguagens são idênticas?) *Sejam L_1 e L_2 duas linguagens regulares quaisquer. Então, a questão $L_1 = L_2$ é decidível.*

Justificativa Considerem-se as linguagens $L_1 = L_1(M_1) \subseteq \Sigma_1^*$ e $L_2 = L_2(M_2) \subseteq \Sigma_2^*$. A condição $L_1 = L_2$ pode também ser formulada como (ver Teorema 1.2):

$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \varnothing$$

onde as operações de complementação se referem a qualquer alfabeto Σ tal que $(\Sigma_1 \cup \Sigma_2) \subseteq \Sigma$. Para decidir se $L_1 = L_2$, deve-se executar o Algoritmo 3.22.

Algoritmo 3.22 (As linguagens são idênticas?) *Determina se duas linguagens regulares são idênticas.*

- Entrada: dois autômatos finitos $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$ e $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$;
- Saída: Se $L_1(M_1) = L_2(M_2)$, SIM; caso contrário, NÃO;
- Método:

    1. Basta construir M_3 tal que $L_3(M_3) = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$. Se $L_3 = \varnothing$, então SIM; caso contrário, NÃO

A construção de M_3 pode ser efetuada diretamente a partir dos algoritmos utilizados na apresentação de resultados anteriores (fechamento dos conjuntos regulares em relação às operações de união, complementação e intersecção, respectivamente Teoremas 3.16, 3.17 e 3.18). Além disso, a questão $L_3(M_3) = \varnothing$ pode ser decidida em função do Teorema 3.23. ■

Teorema 3.28 (A linguagem é Σ*?) *Seja M um autômato que aceita L sobre Σ. Então, a questão $L = \Sigma^*$ é decidível.*

Justificativa Esta questão pode ser decidida pelo Algoritmo 3.23.

Algoritmo 3.23 (A linguagem é Σ*?) *Determina se a linguagem aceita por um autômato finito é Σ^*.*

- Entrada: um autômato finito $M = (Q, \Sigma, \delta, q_0, F)$;
- Saída: Se $L(M) = \Sigma^*$, SIM; caso contrário, NÃO;
- Método:

    1. Basta construir M' tal que $L(M') = \Sigma^* - L(M) = \overline{L(M)}$. Se $L(M') = \varnothing$, então SIM; caso contrário, NÃO.

A validade deste algoritmo e, conseqüentemente, a correção desta demonstração decorrem de resultados anteriormente demonstrados. ■

Teorema 3.29 (A linguagem é subconjunto?) *Sejam $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$ duas linguagens regulares. Então, a questão $L_1 \subseteq L_2$ é decidível.*

Justificativa A condição $L_1 \subseteq L_2$ também pode ser formulada como:

$$(\Sigma^* - L_2) \cap L_1 = \overline{L_2} \cap L_1 = \varnothing$$

onde a operação de complementação se refere a um alfabeto Σ tal que $\Sigma_2 \subseteq \Sigma$. Para decidir se $L_1 \subseteq L_2$, basta executar o Algoritmo 3.24.

Algoritmo 3.24 (A linguagem é subconjunto?) *Determina se uma linguagem regular é subconjunto de uma outra linguagem regular.*

- Entrada: dois autômatos finitos $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$ e $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$;
- Saída: Se $L_1(M_1) \subseteq L_2(M_2)$, SIM; caso contrário, NÃO;
- Método:

    1. Basta construir M_3 tal que $L_3(M_3) = \overline{L_2(M_2)} \cap L_1(M_1)$. Se $L_3 = \varnothing$, então SIM; caso contrário, NÃO.

Também neste caso a validade do algoritmo e da demonstração decorrem de resultados anteriormente apresentados. ■

4 Linguagens Livres de Contexto

As linguagens e as gramáticas livres de contexto foram inicialmente concebidas com a intenção de permitir a formalização sintática das linguagens naturais ([46], [53]). Logo se percebeu, no entanto, que as linguagens naturais (inglês, português etc.) são significativamente mais complexas do que a classe de linguagens representáveis através das gramáticas livres de contexto, diminuindo em muito, em conseqüência, o interesse dos estudiosos das linguagens naturais pelas gramáticas desse tipo.

Por outro lado, as linguagens livres de contexto despertaram um interesse muito grande na comunidade científica ligada à área de computação, que via grandes possibilidades de aplicação dessa classe de linguagens na análise e na formalização de linguagens artificiais, em particular as linguagens de programação.

Vale lembrar que, à época em que isso aconteceu, em meados da década de 1950, a linguagem FORTRAN já havia sido definida, e algumas implementações experimentais estavam sendo utilizadas no desenvolvimento de aplicações. Como a teoria de linguagens ainda principiava o seu desenvolvimento, não surpreende o fato de que a linguagem FORTRAN tenha sido originalmente especificada — e implementada — de maneira quase que totalmente empírica, sem que houvesse, como há hoje, um sólido lastro teórico e conceitual para fundamentar seu projeto.

Por esse motivo, inúmeras dificuldades foram enfrentadas no início do projeto FORTRAN, as quais tiveram como conseqüência os seguintes "registros de época":

1. Elevada "complexidade aparente" dos problemas de formalização sintática e de construção dos aceitadores sintáticos das linguagens de programação.

2. Arbitrariedade na imposição de restrições nos projetos de linguagens de alto nível, tornando mais difícil sua utilização por parte dos usuários.

Como conseqüência das limitações teóricas verificadas na época, praticamente inexistiam métodos para a formalização sintática das linguagens de programação e para a construção sistemática de seus reconhecedores. O advento da teoria das linguagens e seu posterior relacionamento com a teoria dos autômatos ocorreram justamente no momento em que a necessidade de métodos e técnicas havia se tornado bastante clara, em função dessa primeira experiência, a fim de possibilitar a evolução das novas áreas de conhecimento emergentes dentro da computação: a das linguagens de programação e a dos compiladores.

Nesse sentido, o papel das gramáticas livres de contexto foi extremamente importante, uma vez que, se de um lado permitiu o desenvolvimento de notações bastante adequadas para a formalização da sintaxe das linguagens artificiais, por outro ensejou o desenvolvimento de técnicas muito eficientes de construção de reconhecedores sintáticos, algumas delas inclusive através de mapeamentos efetuados diretamente a partir das gramáticas das respectivas linguagens.

Assim, as gramáticas livres de contexto contribuíram significativamente para desmitificar a complexidade dos então novos problemas que se apresentavam, simplificando-os consideravelmente. Um exemplo disso foi a definição da notação BNF ([31]), logo após a

definição das gramáticas livres de contexto por Chomsky ([61]). A notação BNF (sigla de *Backus-Naur Form*) era também destinada à representação de gramáticas livres de contexto e foi concebida especificamente para uso em linguagens artificiais, em particular em linguagens de programação. Ela foi utilizada pela primeira vez na especificação da linguagem de programação Algol 60 ([32], [33]), tendo seu uso se difundido bastante até os dias de hoje.

Atualmente, graças ao grande desenvolvimento teórico verificado nas últimas décadas, tais problemas, longe de terem se tornado triviais, tiveram bem determinados os limites de sua complexidade, e, como conseqüência, a diversificação de métodos e resultados teóricos conhecidos permite que os mesmos possam ser tratados de forma rotineira e sistemática no dia-a-dia do profissional de computação.

Neste capítulo será mostrada a importância das gramáticas livres de contexto na formalização das linguagens de programação de alto nível e serão definidos os conceitos de árvores de derivação e ambigüidade, entre outros. A seguir, serão apresentados e demonstrados alguns importantes resultados referentes à simplificação das gramáticas livres de contexto e à sua representação em formas normais.

4.1 Gramáticas Livres de Contexto

As gramáticas livres de contexto podem ser utilizadas, entre outras aplicações, para representar subconjuntos (bastante limitados) de linguagens naturais. Considere o seguinte exemplo:

$$
\begin{aligned}
\langle \textit{frase} \rangle &\rightarrow \langle \textit{sujeito} \rangle \langle \textit{verbo} \rangle \langle \textit{complemento} \rangle \\
\langle \textit{sujeito} \rangle &\rightarrow \texttt{O homem} \\
\langle \textit{sujeito} \rangle &\rightarrow \texttt{A mulher} \\
\langle \textit{verbo} \rangle &\rightarrow \texttt{leu} \\
\langle \textit{verbo} \rangle &\rightarrow \texttt{escreveu} \\
\langle \textit{complemento} \rangle &\rightarrow \texttt{um} \ \langle \textit{adjetivo} \rangle \ \texttt{livro} \\
\langle \textit{adjetivo} \rangle &\rightarrow \texttt{ótimo} \\
\langle \textit{adjetivo} \rangle &\rightarrow \texttt{péssimo} \\
\langle \textit{adjetivo} \rangle &\rightarrow \epsilon
\end{aligned}
$$

Como se pode perceber, os símbolos não-terminais em gramáticas livres de contexto podem ser utilizados para representar classes sintáticas bem definidas, permitindo um elevado grau de intuição e de clareza na interpretação e na estruturação das sentenças pertencentes à linguagem.

Quando utilizadas para representar subconjuntos de linguagens naturais, ou mesmo linguagens "pseudonaturais", como, por exemplo, as linguagens empregadas em sistemas de consulta a bancos de dados e em diversos outros tipos de interfaces homem-máquina, especialmente nos aplicativos voltados para usuários leigos em computação, as gramáticas livres de contexto desempenham um papel muito importante para os projetistas de tais sistemas, pois permitem a formalização da linguagem de interface, facilitando a construção de seu respectivo reconhecedor.

No entanto, a maior aplicação das gramáticas livres de contexto ocorre na formalização sintática das linguagens de programação de alto nível. Rigorosamente, não se

pode dizer que tais linguagens sejam propriamente livres de contexto. Na verdade, a formalização completa de sua sintaxe exigiria dispositivos mais complexos, como é o caso das gramáticas do tipo 1, uma vez que a grande maioria de tais linguagens apresenta dependências de contexto que não podem ser representadas por gramáticas do tipo 2, ou livres de contexto.

Não obstante, a simplicidade das gramáticas livres de contexto as torna muito atraentes para a representação formal, ainda que parcial, da sintaxe de tais linguagens. As gramáticas livres de contexto constituem, pois, uma solução de compromisso em que o rigor da definição sintática é ligeiramente sacrificado em favor do uso de uma notação muito mais intuitiva e fácil de ser manipulada do que as gramáticas do tipo 1. Desse modo, as dependências de contexto costumam ser representadas à parte, em alguma outra notação — eventualmente em linguagem natural — que sirva de complementação à gramática do tipo 2 para a representação das dependências de contexto. As gramáticas formais do tipo 1, devido à sua elevada complexidade de manipulação, não costumam ser utilizadas na prática para representar dependências de contexto em linguagens de programação, nem mesmo de forma parcial.

A característica que torna as gramáticas livres de contexto especialmente adequadas à formalização sintática das linguagens de programação é a sua capacidade de representação de **construções aninhadas**, que são freqüentemente encontradas em linguagens dessa categoria. Construções aninhadas costumam ocorrer em linguagens de programação, por exemplo, na construção de expressões aritméticas, em que subexpressões são delimitadas, através do uso de parênteses; na estruturação do fluxo de controle, em que comandos internos são inseridos como parte integrante de outros externos; na estruturação do programa, em que blocos, módulos, procedimentos e funções são empregados para criar diferentes escopos, etc.

Conforme definido no Capítulo 2, uma gramática livre de contexto é uma quádrupla (V, Σ, P, S) com os seguintes componentes:

- V: conjunto (finito e não-vazio) dos símbolos terminais e não-terminais;
- Σ: conjunto (finito e não-vazio) dos símbolos terminais; corresponde ao alfabeto da linguagem definida pela gramática;
- P: conjunto (finito e não-vazio) das regras de produção, todas no formato $\alpha \rightarrow \beta$, com $\alpha \in (V - \Sigma)$ e $\beta \in V^*$;
- S: raiz da gramática, $S \in (V - \Sigma)$.

Exemplo 4.1 Seja P o conjunto de regras abaixo e considerem-se Σ, V e S subentendidos.

$$\begin{aligned}
\{E &\rightarrow T + E, && (4.1)\\
E &\rightarrow T, && (4.2)\\
T &\rightarrow F * T, && (4.3)\\
T &\rightarrow F, && (4.4)\\
F &\rightarrow (E), && (4.5)\\
F &\rightarrow a\} && (4.6)
\end{aligned}$$

Considere-se a sentença $a * (a + a)$. Ela pode ser obtida através da seguinte seqüência de derivações:

$E \Rightarrow T \Rightarrow F * T \Rightarrow a * T \Rightarrow a * F \Rightarrow a * (E) \Rightarrow a * (T + E) \Rightarrow a * (F + E) \Rightarrow a * (a + E)$
$\Rightarrow a * (a + T) \Rightarrow a * (a + F) \Rightarrow a * (a + a)$

correspondente à aplicação das produções $4.2, 4.3, 4.6, 4.4, 4.5, 4.1, 4.4, 4.6, 4.2, 4.4, 4.6$, nesta ordem.

Observe-se que a linguagem gerada pela gramática deste exemplo compreende as sentenças que representam expressões aritméticas corretamente formadas sobre o operando a com os operadores "$*$" e "$+$". Subexpressões delimitadas através de parênteses também são admitidas, e podem ser compostas com base nas mesmas regras utilizadas para construir a expressão inicial. Não fazem parte da linguagem definida por esta gramática, por exemplo, cadeias em que não haja plena correspondência do símbolo "(" com seu par ")". Em outras palavras, trata-se de uma linguagem que admite o aninhamento de expressões através do uso de parênteses como delimitadores. □

O aninhamento de construções é a característica que distingue as linguagens estritamente do tipo 2 das linguagens do tipo 3: pertencem a uma linguagem estritamente do tipo 2 apenas e tão somente as sentenças em que, para cada ocorrência de um dado delimitador (no caso, o "abre-parênteses"), haja em correspondência a ocorrência de um outro, através do qual se forma o par (no caso, o "fecha-parênteses"). Além disso, a subcadeia situada entre esse par de delimitadores pode ser gerada através das mesmas regras de formação válidas para a sentença completa, caracterizando dessa maneira o aninhamento de suas construções.

Note-se que, através de regras de produção livres de contexto, é possível especificar construções mais restritas do que as obteníveis mediante o uso de produções de gramáticas lineares. Assim, com regras livres de contexto é possível caracterizar aninhamentos sintáticos, os quais não podem ser descritos apenas com produções lineares à direita ou à esquerda (ou qualquer combinação delas). Em outras palavras, as gramáticas livres de contexto permitem impor restrições adicionais àquelas que se podem construir em gramáticas regulares, podendo assim caracterizar subconjuntos das linguagens regulares que gozem da propriedade determinada pelos aninhamentos sintáticos. Dessa forma, as gramáticas livres de contexto tornam-se muito úteis para a especificação de linguagens de programação, a maioria das quais exibe aninhamentos sintáticos.

A representação de aninhamentos em gramáticas do tipo 2 decorre de uma propriedade característica das gramáticas deste tipo, denominada "Self-embedding Property". Um não-terminal Y é dito "self-embedded", ou **auto-recursivo central**, se, a partir dele, for possível derivar alguma forma sentencial em que o não-terminal Y ressurja, delimitado por cadeias não-vazias de terminais à sua esquerda e à sua direita:

$$Y \Rightarrow^* \alpha Y \beta, \quad \text{com } \alpha, \beta \in \Sigma^+$$

Um não-terminal Z é dito simplesmente **auto-recursivo** se, a partir dele, for possível derivar alguma forma sentencial em que Z ressurja, acompanhado de pelo menos uma cadeia não-vazia de terminais à sua esquerda ou à sua direita:

$$Z \Rightarrow \alpha Z \beta, \quad \text{com } \alpha, \beta \in \Sigma^*, \alpha\beta \neq \epsilon$$

Se uma gramática livre de contexto G possui pelo menos um não-terminal auto-recursivo central, diz-se que G é **auto-embutida** (do inglês "self-embedded").

Um símbolo não-terminal **essencial** é aquele que não pode ser removido da gramática (ou substituído) sob pena de provocar modificações na linguagem sendo definida. Uma linguagem L é dita **estritamente livre de contexto**, ou livre de contexto não-regular, se e apenas se todas as gramáticas que geram L forem auto-embutidas, ou seja, se todas elas possuírem pelo menos um não-terminal **auto-recursivo central essencial**.

O simples fato de uma gramática ser auto-embutida não garante a não-regularidade da linguagem definida: é possível identificar linguagens regulares geradas por gramáti-

cas com não-terminais auto-recursivos centrais que, nesses casos, não são essenciais. O Exemplo 4.2 ilustra essa situação.

Exemplo 4.2 A gramática cujas regras constituem o conjunto abaixo é do tipo 2 e possui um não-terminal auto-recursivo central (S, em decorrência da produção $S \rightarrow aSa$), podendo portanto ser caracterizada como uma gramática auto-embutida:

$$\begin{array}{rcl} \{S & \rightarrow & aS, \\ S & \rightarrow & bS, \\ S & \rightarrow & a, \\ S & \rightarrow & b, \\ S & \rightarrow & aSa\} \end{array}$$

No entanto, a linguagem gerada por essa gramática é $\{a, b\}^*$, ou seja, a linguagem é regular. Na verdade, é fácil observar que essa linguagem também pode ser gerada por um conjunto de regras equivalente, em que a última produção da gramática acima é removida:

$$\begin{array}{rcl} \{S & \rightarrow & aS, \\ S & \rightarrow & bS, \\ S & \rightarrow & a, \\ S & \rightarrow & b\} \end{array}$$

Tal fato ocorre, neste caso particular, porque a produção $S \rightarrow aSa$ é **não-essencial** à gramática, ou seja, a sua inclusão no conjunto P de produções em nada contribui para modificar a linguagem definida pelas demais produções. □

Quando todas as alternativas de substituição para um símbolo não-terminal são não-essenciais, diz-se que o símbolo em questão é **não-essencial**. Se ele for um não-terminal auto-recursivo central, diz-se que o não-terminal auto-recursivo central é não-essencial. Dessa maneira, os não-terminais auto-recursivos centrais que caracterizam gramáticas auto-embutidas podem ou não ser essenciais. Se houver pelo menos um não-terminal auto-recursivo central essencial em alguma gramática, a correspondente linguagem é dita **estritamente livre de contexto**. Se todos os não-terminais auto-recursivos centrais de uma gramática forem não-essenciais, a linguagem por ela definida é regular.

A "Self-embedding Property" exprime a capacidade que têm as gramáticas livres de contexto para representarem aninhamentos sintáticos. Intuitivamente, se $Y \Rightarrow^* \alpha Y \beta$, então, da aplicação sucessiva dessa seqüência de derivações, resulta que $\alpha\alpha Y \beta\beta$, $\alpha\alpha\alpha Y \beta\beta\beta$ etc. são formas sentenciais que geram sentenças pertencentes a $L(G)$. Assim, a cada subcadeia α é possível impor a existência de uma outra subcadeia β que estabeleça a correspondência com α. É importante notar, finalmente, que as gramáticas regulares não exibem a "Self-embedding Property", sendo esta uma propriedade característica das gramáticas livres de contexto não-regulares.

Exemplo 4.3 Considere-se a linguagem das expressões aritméticas definida no Exemplo 4.1. Observe-se que $E \Rightarrow T \Rightarrow F \Rightarrow (E)$, ou seja, $E \Rightarrow^* (E)$.

Logo, E é um não-terminal auto-recursivo central, e a gramática a que ele pertence é auto-embutida. É possível demonstrar que todas as gramáticas que geram essa linguagem são auto-embutidas, o que caracteriza a linguagem como não-regular. Note-se que, como $E \Rightarrow^* (E)$, a cada abre-parênteses que antecede a subexpressão E corresponde sempre um fecha-parênteses logo após E. Note-se também que todas as sentenças derivadas das formas sentenciais $({}^iE)^i$, $i \geq 1$, também pertencem à linguagem definida, sendo obtidas pela aplicação repetida da seqüência de derivações apresentada. □

Quando se consideram linguagens especificadas através de gramáticas livres de contexto, deve-se também considerar de que forma é feita a aceitação sintática de suas sentenças para fins de compilação e/ou interpretação. Quando se trata de efetuar o re-

conhecimento de sentenças, o que se busca, na verdade, é localizar uma seqüência de produções que, quando aplicada à raiz da gramática, forneça como resultado, através da série correspondente de derivações, a sentença fornecida para análise. Sendo possível completar a derivação, diz-se que a sentença pertence à linguagem; caso contrário, que ela não pertence à linguagem.

No entanto, para cada sentença pertencente à linguagem definida através de uma gramática livre de contexto, é geralmente possível identificar uma grande quantidade de seqüências distintas de derivação, resultado de escolhas arbitrárias do particular símbolo não-terminal a ser substituído em cada passo da derivação, todas elas resultando na mesma sentença analisada. Na prática, no entanto, costuma-se fixar alguns critérios de derivação de sentenças, para permitir a construção e a operação sistemáticas dos reconhecedores sintáticos.

Assim, diz-se que a derivação de uma forma sentencial em uma gramática livre de contexto é uma **derivação mais à esquerda**, quando a substituição de um não-terminal pelo lado direito de uma produção que o define é feita substituindo-se sempre o não-terminal que ocorre mais à esquerda na cadeia que representa a forma sentencial em questão. De maneira análoga, uma **derivação mais à direita** é aquela em que é sempre o não-terminal mais à direita, na forma sentencial, que é substituído pela sua definição.

Exemplo 4.4 Considere-se a gramática do Exemplo 4.3. A seguir são apresentadas duas seqüências de derivação para a sentença $a + a$. Na primeira, todas as derivações são mais à esquerda, e na segunda, mais à direita. Note-se que as seqüências de produções utilizadas em cada caso são distintas, e também que diversas outras seqüências poderiam ser obtidas combinando-se arbitrariamente os não-terminais a serem substituídos em cada forma sentencial:

1. Derivações com substituições mais à esquerda apenas:
   $E \Rightarrow T + E \Rightarrow F + E \Rightarrow a + E \Rightarrow a + T \Rightarrow a + F \Rightarrow a + a$
   Produções aplicadas: $4.1, 4.4, 4.6, 4.2, 4.4, 4.6$
2. Derivações com substituições mais à direita apenas:
   $E \Rightarrow T + E \Rightarrow T + T \Rightarrow T + F \Rightarrow T + a \Rightarrow F + a \Rightarrow a + a$
   Produções aplicadas: $4.1, 4.2, 4.4, 4.6, 4.4, 4.6$
3. Derivações com substituições de diversos tipos:
   $E \Rightarrow T + E \Rightarrow F + E \Rightarrow F + T \Rightarrow a + T \Rightarrow a + F \Rightarrow a + a$
   Produções aplicadas: $4.1, 4.4, 4.2, 4.6, 4.4, 4.6$

□

Como conseqüência do elevado interesse prático despertado pelas gramáticas livres de contexto, inúmeras notações foram desenvolvidas para facilitar a formalização sintática das linguagens artificiais, permitindo criar definições formais mais legíveis e concisas do que as obtidas usualmente com a notação algébrica.

Tais notações, denominadas **metalinguagens**, permitem a representação de linguagens livres de contexto, sendo equipotentes, portanto, à notação algébrica introduzida na Seção 2.3. A primeira e talvez mais importante delas é a BNF, abreviatura de "Backus-Naur Form".

Na notação BNF, os não-terminais são representados por textos delimitados pelos metasímbolos "<" e ">"; para distingui-los dos símbolos terminais, o metasímbolo "→" é substituído por "::=" e, finalmente, todas as alternativas de substituição para um mesmo não-terminal são agrupadas, separando-se umas das outras com o metasímbolo "|". Os terminais são denotados sem delimitadores.

Exemplo 4.5 Em BNF, a gramática que representa expressões aritméticas aninháveis, sobre operandos a, com os operadores adição e multiplicação, anteriormente apresentada no Exemplo 4.1, torna-se:

$$\begin{array}{lcl} \langle E \rangle & ::= & \langle T \rangle + \langle E \rangle \mid \langle T \rangle \\ \langle T \rangle & ::= & \langle F \rangle * \langle T \rangle \mid \langle F \rangle \\ \langle F \rangle & ::= & a \mid (\langle E \rangle) \end{array}$$

□

Empregada pela primeira vez no início da década de 1960, na publicação que definiu formalmente a linguagem de programação Algol 60 ([32], [33]), a notação BNF se tornou extremamente popular justamente por ter demonstrado na prática, pela primeira vez, a viabilidade de uso de uma notação formal para a representação da sintaxe de uma linguagem de programação, representando um dos primeiros resultados práticos advindos do interesse dos profissionais de computação pelas então recém descobertas gramáticas livres de contexto.

Embora largamente utilizada ainda hoje, a notação BNF enfrenta atualmente a "concorrência" de outras metalinguagens igualmente importantes, como é o caso de vários de seus dialetos, da Notação de Wirth, dos Diagramas de Sintaxe, das Expressões Regulares Estendidas e de inúmeras outras ([24]). Apesar disso, a notação algébrica continuará sendo empregada preferencialmente neste texto por ser a mais adequada ao estudo teórico e conceitual das linguagens formais. Eventualmente, poderá ser empregada a BNF, em exemplos de linguagens de programação.

4.2 BNF Estendida

Expressões regulares foram definidas como uma notação bastanta concisa e adequada para a representação de linguagens regulares. Uma importante extensão das expressões regulares são as expressões regulares estendidas, que, em combinação com a BNF, disponibilizam as vantagens do uso de expressões regulares para a classe das linguagens livres de contexto, constituindo assim uma metalinguagem alternativa para a representação das mesmas.

Essa metalinguagem, denominada **BNF estendida** (ou **EBNF**), resulta da fusão das definições da BNF (conforme a Seção 4.1) e da expressão regular (conforme a Seção 3.2).

Uma **expressão regular estendida** é, por definição, uma expressão regular que admite como operandos os símbolos não-terminais da gramática, em adição aos terminais.

Um conjunto de regras gramaticais representado através da notação **BNF estendida** é um conjunto de expressões regulares estendidas, cada uma delas associada a um símbolo não-terminal distinto.

Exemplo 4.6 Considere-se a gramática livre de contexto:

$$G = (\{S, X, Y, Z, a, b, c, d, e, f, g\}, \{a, b, c, d, e, f, g\}, P, S)$$

com P apresentado a seguir, que gera uma linguagem L estritamente livre de contexto.

$$\begin{array}{rcl} P = \{S & \rightarrow & XYZ \mid g \\ X & \rightarrow & aX \mid a \\ Y & \rightarrow & Sb \\ Z & \rightarrow & cdZ \mid eZ \mid f\} \end{array}$$

Transcrito para a BNF, este conjunto de produções resulta:

$$\begin{aligned} \langle S\rangle &::= \langle X\rangle\langle Y\rangle\langle Z\rangle \mid g \\ \langle X\rangle &::= a\langle X\rangle \mid a \\ \langle Y\rangle &::= \langle S\rangle b \\ \langle Z\rangle &::= cd\langle Z\rangle \mid e\langle Z\rangle \mid f \end{aligned}$$

Não é difícil perceber que <X> gera cadeias compostas por um ou mais símbolos a. Então, a regra $\langle X\rangle ::= a\langle X\rangle \mid \epsilon$ pode ser substituída pela regra $\langle X\rangle ::= aa^*$. De maneira análoga, a regra $\langle Z\rangle ::= cd\langle Z\rangle \mid e\langle Z\rangle \mid f$ pode ser substituída por $\langle Z\rangle ::= (cd \mid e)^*f$. Note-se, em ambos os casos, a substituição do uso de símbolo não-terminal no lado direito das regras, pelo uso do operador fechamento reflexivo e transitivo ($*$) para representar a repetição de termos. O novo conjunto de regras torna-se, portanto:

$$\begin{aligned} \langle S\rangle &::= \langle X\rangle\langle Y\rangle\langle Z\rangle \mid g \\ \langle X\rangle &::= aa^* \\ \langle Y\rangle &::= \langle S\rangle b \\ \langle Z\rangle &::= (cd \mid e)^*f \end{aligned}$$

A substituição das definições dos símbolos <X>, <Y> e <Z> na regra do símbolo <S> resulta em:

$$\langle S\rangle ::= aa^*\langle S\rangle b(cd \mid e)^*f \mid g$$

O lado direito da regra acima ($aa^*\langle S\rangle b(cd \mid e)^*f \mid g$) é, como se pode perceber, uma expressão regular estendida, pois possui a forma de uma expressão regular, acrescida da referência ao símbolo não-terminal <S>. O lado esquerdo e o lado direito, juntamente, constituem uma regra gramatical representada na notação BNF estendida.

A regra acima explicita a presença de um símbolo não-terminal auto-recursivo essencial em G (no caso, o símbolo <S>), suficiente para caracterizar L como sendo livre de contexto e não-regular. Em termos informais, L pode também ser representada como $(aa^*)^n g(b(cd \mid e)^*f)^n$, com $n \geq 0$.□

A principal vantagem decorrente uso da BNF estendida, em comparação com a notação algébrica, ou mesmo com a BNF, resulta da possibilidade de representar a repetição de formas sintáticas sem a necessidade de definições gramaticais recursivas (aquelas em que o símbolo não-terminal que estiver sendo definido ressurge, direta ou indiretamente, em formas sentenciais derivadas do mesmo), substituindo-as pela definição de iterações explícitas (através do uso do operador fechamento reflexivo e transitivo). Isso proporciona um entendimento mais fácil da linguagem por ela definida, sendo ainda útil em determinados métodos de construção de reconhecedores sintáticos a partir de gramáticas livres de contexto (ver Seção **??**).

Note-se, no entanto, que nem sempre uma gramática livre de contexto poderá ser reduzida a uma única regra em BNF estendida, ainda que esta represente uma definição recursiva. De fato, a quantidade mínima de regras a que se pode chegar é igual à quantidade de símbolos não-terminais auto-recursivos centrais essenciais que a gramática possui, uma vez que cada um destes é responsável pela incorporação de uma característica distinta de aninhamento sintático à linguagem definida pela gramática, não podendo ser removidos da gramática sem prejuízo da linguagem por ela definida.

Por outro lado, gramáticas livres de contexto que representam linguagens regulares poderão sempre ser manipuladas até a eliminação de todos os símbolos não-terminais da gramática, exceto, naturalmente, a raiz, resultando em uma única expressão regular

(não-estendida) que gera a linguagem definida pela gramática. Dessa maneira, a BNF estendida é um formalismo que une as vantagens das expressões regulares às da BNF, constituindo importante alternativa tanto para a representação de linguagens regulares quanto para a de linguagens livres de contexto.

Diz-se que um símbolo não-terminal A de uma gramática livre de contexto G é **recursivo à esquerda** se $A \Rightarrow^* A\alpha$, com $\alpha \in V^*$. Uma gramática é dita **recursiva à esquerda** se ela possuir pelo menos um símbolo não-terminal recursivo à esquerda. De maneira análoga, diz-se que um símbolo não-terminal A de uma gramática livre de contexto G é **recursivo à direita** se $A \Rightarrow^* \alpha A$, com $\alpha \in V^*$. Uma gramática é dita **recursiva à direita** se ela possuir pelo menos um símbolo não-terminal recursivo à direita.

Observe-se que recursões à esquerda podem ocorrer de forma direta — através de produções do tipo $A \rightarrow A\alpha$, $\alpha \in V^*$ — ou indireta — como, por exemplo, em $A \rightarrow B\alpha$, $B \rightarrow C\beta$ e $C \rightarrow A\gamma$, pois $A \Rightarrow B\alpha \Rightarrow C\beta\alpha \Rightarrow A\gamma\beta\alpha$, ou seja, $A \Rightarrow^* A\phi$, com $\phi = (\gamma\beta\alpha)^*$. Considerações semelhantes são válidas para recursões à direita.

Exemplo 4.7 Considere-se a gramática cujas regras são:

$$\begin{array}{rcl} \langle S\rangle & ::= & a\langle S\rangle \mid b\langle X\rangle \\ \langle X\rangle & ::= & \langle X\rangle b \mid c\langle S\rangle \mid \langle Y\rangle bc \\ \langle Y\rangle & ::= & \langle X\rangle d \mid \langle Z\rangle \mid e \\ \langle Z\rangle & ::= & \langle X\rangle f \mid \langle W\rangle \\ \langle W\rangle & ::= & e\langle W\rangle f \mid g \end{array}$$

Neste caso, é possível identificar as seguintes situações: o símbolo não-terminal <S> é recursivo à direita diretamente (através da regra <S> ::= a<S>), e indiretamente (através das regras <S> ::= b<X>, <X> ::= c<S>). O símbolo não-terminal <X>, por sua vez, é recursivo à esquerda diretamente (através da regra <X> ::= <X>b) e indiretamente (através das regras <X> ::= <Y>bc, <Y> ::= <X>d, <Y> ::= <Z>, <Z> ::= <X>f). Note-se que o símbolo <W> não é recursivo à esquerda nem à direita. □

O lado direito de uma regra $A \rightarrow \alpha, \alpha \in V^*$, em uma gramática livre de contexto, é dito **termo independente de** A, se A não for uma subcadeia de α.

Seja G uma gramática livre de contexto, e $A \in N$ um símbolo não-terminal de G. Então, as regras de A podem exibir um ou mais dos seguintes formatos gerais (considera-se $\alpha, \beta, \gamma, \mu$, e $\rho \in V^*$, e $\delta \in (V - \{A\})^*$):

- recursão à direita: $A \rightarrow \alpha A$;
- recursão à esquerda: $A \rightarrow A\beta$;
- recursão à esquerda e à direita: $A \rightarrow A\gamma A$;
- recursão central: $A \rightarrow \mu\, A\rho$;
- termo independente: $A \rightarrow \delta$.

Agrupando-se as regras de mesmo formato, obtém-se:

$$\begin{array}{rcl} A & \rightarrow & \alpha_1 A \mid \alpha_2 A \mid ... \mid \alpha_i A \mid \\ & & A\beta_1 \mid A\beta_2 \mid ... \mid A\beta_j \mid \end{array}$$

$$A\gamma_1 A \mid A\gamma_2 A \mid ... \mid A\gamma_k A \mid$$
$$\mu_1\, A\rho_1 \mid \mu_2\, A\rho_2 \mid ... \mid \mu_m\, A\rho_m \mid$$
$$\delta_1 \mid \delta_2 \mid ... \mid \delta_p$$

É possível demonstrar, conforme discussão a seguir, que a fórmula geral que permite representar a linguagem gerada por A, sem no entanto fazer uso de recursões à esquerda e/ou à direita, é:

$$\begin{aligned} A \;=\; & \\ & (\alpha_1 \mid ... \mid \alpha_i)^*(\delta_1 \mid ... \mid \delta_p \mid \mu_1\, A\rho_1 \mid ... \mid \mu_m\, A\rho_m)(\beta_1 \mid ... \mid \beta_j)^* \\ & (\\ & (\gamma_1 \mid ... \mid \gamma_k) \\ & (\alpha_1 \mid ... \mid \alpha_i)^*(\delta_1 \mid ... \mid \delta_p \mid \mu_1\, A\rho_1 \mid ... \mid \mu_m\, A\rho_m)(\beta_1 \mid ... \mid \beta_j)^* \\ &)^* \end{aligned}$$

As recursões centrais não podem, naturalmente, ser eliminadas na expressão resultante, sob pena de se modificar a linguagem definida por A, no caso das linguagens estritamente livres de contexto. O mesmo vale para o caso de múltiplas recursões em uma mesma regra, que não sejam da forma $A\gamma A$ (por exemplo, como em $A \rightarrow A\sigma_1 A\sigma_2 A\sigma_3 A$).

Se $i = j = k = m = p = 1$, o caso geral reduz-se a:

$$A \;\rightarrow\; \alpha A \mid A\beta \mid A\gamma A \mid \mu\, A\rho \mid \delta$$

cuja solução é:

$$A \;=\; \alpha^*(\delta \mid \mu\, A\rho)\beta^*(\gamma\alpha^*(\delta \mid \mu\, A\rho)\beta^*)^*$$

Nesta fórmula, constatam-se as seguintes relações de causa e efeito (ver Exercício **??** da Seção **??**):

- As cadeias geradas por A são iniciadas com uma repetição arbitrária do termo α, decorrente das recursões à direita existentes na gramática;
- As cadeias geradas por A são terminadas com uma repetição arbitrária do termo β, decorrente das recursões à esquerda existentes na gramática;
- Os termos independentes (δ) e as recursões centrais ($\mu\, A\rho$) são preservados no centro das cadeias geradas por A, isto é, à direita dos termos α e à esquerda dos termos β;
- As regras que são simultaneamente recursivas à esquerda e à direita permitem a concatenação de múltiplas cadeias geradas por A, tendo o termo γ como separador entre elas.

Exemplo 4.8 Alguns casos simples, envolvendo apenas recursões à esquerda, à direita, e termos independentes, são apresentados a seguir.

- Se $X \rightarrow aX \mid \epsilon$, então $X = a^*$;

- Se $X \rightarrow aX \mid b$, então $X = a^*b$;
- Se $X \rightarrow Xb \mid a$, então $X = ab^*$;
- Se $X \rightarrow Xb \mid \epsilon$, então $X = b^*$;
- Se $X \rightarrow aX \mid bX \mid c$, então $X = (a \mid b)^*c$;
- Se $X \rightarrow Xa \mid Xb \mid c$, então $X = c(a \mid b)^*$;
- Se $X \rightarrow aX \mid b \mid c$, então $X = a^*(b \mid c)$;
- Se $X \rightarrow aX \mid Xb \mid \epsilon$, então $X = a^*b^*$;
- Se $X \rightarrow aX \mid Xc \mid b$, então $X = a^*bc^*$;
- Se $X \rightarrow aX \mid Xd \mid b \mid c$, então $X = a^*(b \mid c)d^*$;
- Se $X \rightarrow aX \mid bX \mid Xe \mid c \mid d$, então $X = (a \mid b)^*(c \mid d)e^*$;
- Se $X \rightarrow aX \mid bX \mid Xe \mid Xf \mid c \mid d$, então $X = (a \mid b)^*(c \mid d)(e \mid f)^*$.

□

Exemplo 4.9 Os casos abaixo contêm, além de recursões à esquerda, à direita e termos independentes, recursões centrais, regras recursivas simultaneamente à esquerda e à direita, e regras com múltiplas recursões em outros formatos. Note-se que, nos casos em que o valor de X pode ser representado na forma de um termo independente, é utilizada a notação "X=..." para indicar a associação de valor ao símbolo. Nos casos em que isso não é possível, utiliza-se a notação EBNF "<X>::=...".

- Se $X \rightarrow XcX \mid e$, então $X = e(ce)^*$;
- Se $X \rightarrow aX \mid XcX \mid e$, então $X = a^*e(ca^*e)^*$;
- Se $X \rightarrow aX \mid Xb \mid XcX \mid e$, então $X = a^*eb^*(ca^*eb^*)^*$;
- Se $X \rightarrow aX \mid Xc \mid XX \mid b$, então $X = a^*bc^*(a^*bc^*)^*$;
- Se $X \rightarrow aX \mid bX \mid cXd \mid e$,
  então <X> ::= $(a \mid b)^*(c$<X>$d \mid e)$;
- Se $X \rightarrow aX \mid Xb \mid cXd \mid XeX \mid f$,
  então <X> ::= $a^*fb^*((e \mid c$<X>$d)a^*fb^*)^*$;
- Se $X \rightarrow aX \mid dX \mid Xd \mid bXcc \mid XbbXc \mid e$,
  então <X> ::= $(a \mid d)^*(e \mid b$<X>$cc)(d \mid bb$<X>$c)^*$;
- Se $X \rightarrow aX \mid Xd \mid bXc \mid XeX \mid XXX \mid f$,
  então <X> ::= $a^*(f \mid b$<X>$c)d^*((e \mid$ <X>$)(a^*(f \mid b$<X>$c)d^*))^*$.

□

A notação EBNF teve origem em um artigo publicado em 1977 ([35]). Nessa publicação, Wirth questionava a grande diversidade de notações para a representação de sintaxe existente naquela época, e propunha uma nova variante da BNF como alternativa a todas elas. Tal notação viria a ser denominada, posteriormente, **Notação de Wirth**.

Outras extensões foram sendo propostas e incorporadas ao longo do tempo, de tal modo que atualmente existem inúmeros dialetos daquilo que conhecemos como BNF estendida. Uma tentativa de padronização surgiu com a publicação de um padrão internacional ISO em 1996 ([36]).

Entre as principais extensões que foram sendo desenvolvidas e eventualmente incorporadas a algum dialeto da BNF estendida, podem-se destacar:

- O uso dos metasímbolos "[" e "]" para agrupar termos opcionais, dispensando assim o uso do símbolo ϵ para representar a cadeia vazia (por exemplo, $S \rightarrow aS \mid \epsilon$ pode ser reescrito como $<S> ::= [a<S>]$);

- O uso dos metasímbolos "{" e "}" para agrupar termos que se repetem zero ou mais vezes, dispensando assim o uso de recursões explícitas e também o uso de parênteses no agrupamento do termo que se repete (por exemplo, $S \rightarrow aS \mid b$, cuja solução é $S = \{a\}b$);

- O uso do metasímbolo "+" para representar a repetição de um termo uma ou mais vezes (por exemplo, $S \rightarrow aS \mid a$, cuja solução é $S = a^+$);

- O emprego de aspas duplas para delimitar os símbolos terminais da gramática (às vezes também o negrito ou o sublinhado), mantendo-se os símbolos não-terminais sem qualquer tipo de destaque nas regras gramaticais (por exemplo, $S ::=$ "a"S, $S ::= \mathbf{a}$S, ou ainda $S ::= \underline{a}S$ para denotar $S \rightarrow aS$),

- O emprego do símbolo "=", como alternativa ao de "::=", na separação das partes esquerda (α) e direita (β) das regras de produção gramaticais ($\alpha \rightarrow \beta$);[1]

- O uso do metasímbolo "." para indicar o término de uma regra ($S \rightarrow aX \mid bY$ torna-se $<S> ::= a<X> \mid b<Y>$.).

Considerada hoje um dialeto da EBNF, a Notação de Wirth será apresentada no Exercício **??** da Seção **??**.

4.3 Árvores de Derivação

A representação da estrutura de sentenças ou formas sentenciais de linguagens livres de contexto, na forma de árvores bidimensionais, é um recurso muito utilizado, tanto na teoria quanto na prática da implementação de linguagens, uma vez que:

1. Proporciona meios para uma melhor visualização da estrutura das sentenças da linguagem, facilitando a análise das mesmas.

2. Auxilia na demonstração formal de teoremas, na interpretação de certos resultados teóricos e na assimilação de vários conceitos.

3. Facilita a representação interna, nos compiladores e interpretadores, da estrutura das sentenças analisadas, registrando importantes informações estruturais sobre as mesmas, a serem utilizadas em outros estágios do processamento da linguagem.

[1] Neste texto, o símbolo "=" é usado para indicar que um símbolo está ligado a um determinado valor. O seu uso no lugar do símbolo "::=", ou mesmo do símbolo "→", é desencorajado, uma vez que estes se empregam na representação de regras gramaticais.

Formalmente, uma **árvore de derivação** é um sistema de representação de seqüências de derivações em uma gramática livre de contexto $G = (V, \Sigma, P, S)$, consistindo em um grafo orientado e ordenado, acíclico, com as seguintes propriedades:

1. Todo vértice é rotulado com um elemento de $V \cup \{\epsilon\}$;
2. O rótulo da raiz é S;
3. Os rótulos de vértices internos são elementos de $N = V - \Sigma$;
4. Se um vértice tem o rótulo A, e $X_1, X_2, \ldots X_n$ são descendentes diretos de A, ordenados da esquerda para a direita, então $A \Rightarrow X_1, X_2, ...X_n$ deve pertencer ao conjunto P de regras;
5. Se um vértice possui o rótulo ϵ, então este vértice deve ser simultaneamente uma folha e descendente único de seu ancestral direto.

Define-se como **fronteira de uma árvore** a cadeia formada pela concatenação dos símbolos correspondentes aos rótulos dos vértices que são folhas, considerados no sentido da esquerda para a direita, e de cima para baixo. Essa cadeia de símbolos (terminais e/ou não-terminais) corresponde à forma sentencial cuja estrutura está sendo representada pela árvore.

Exemplo 4.10 Considere-se a gramática das expressões aritméticas anteriormente apresentada no Exemplo 4.1. A estrutura da sentença $a * (a + a)$, de acordo com essa gramática, pode ser representada pela árvore de derivação:

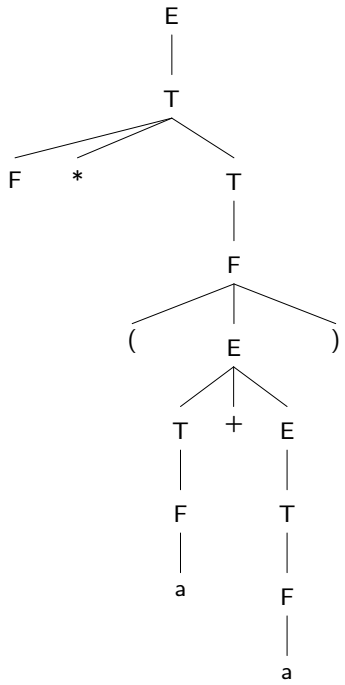

Figura 4.1: Árvore de derivação para $E \Rightarrow^* a * (a + a)$

Observe-se, nesta árvore, que a sentença $a * (a + a)$ pode ser obtida pela concatenação dos rótulos dos vértices que são folhas, e corresponde à sentença cuja estrutura está sendo representada pela própria árvore. □

Naturalmente, nem toda árvore necessita ter como raiz S, nem tampouco possuir como fronteira $\alpha \in \Sigma^*$, $S \Rightarrow^* \alpha$. É comum que se considerem subárvores cuja raiz seja um elemento X qualquer, $X \in N$, e a fronteira, uma cadeia $\gamma \in V^*$, tal que $X \Rightarrow^* \gamma$.

Exemplo 4.11 A derivação não-trivial $T \Rightarrow^* a^*(E)$, segundo a gramática do Exemplo 4.1, pode ser representada através da subárvore:

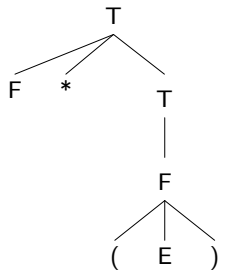

Figura 4.2: Árvore de derivação para $T \Rightarrow^* a * (a + a)$

□

A seguir, será apresentado o teorema fundamental das árvores de derivação. Ele estabelece a existência de pelo menos uma árvore de derivação para toda e qualquer cadeia derivável a partir da raiz da gramática, e também que toda e qualquer fronteira de uma árvore corretamente construída sobre uma gramática livre de contexto G corresponde a uma cadeia derivável a partir da raiz de G.

Teorema 4.1 (Árvores $\Leftrightarrow$ derivações) *Seja $G = (V, \Sigma, P, S)$ uma gramática livre de contexto. Então $S \Rightarrow^* \alpha$ se e apenas se existir uma árvore de derivação sobre G com fronteira α.*

Justificativa Por se tratar de um teorema bastante intuitivo, não será apresentada a sua prova formal, que no entanto pode ser encontrada em [46]. A importância deste teorema se deve ao fato de que ele "credencia" as árvores de derivação como uma representação válida da estrutura das formas sentenciais geradas por uma gramática. ■

Árvores de derivação não contêm informação sobre a particular seqüência em que foram aplicadas as produções para a obtenção de uma dada forma sentencial: elas informam apenas **quais** foram as produções aplicadas, mas não **em que ordem**. Assim, para cada árvore de derivação, é possível identificar diversas seqüências distintas de derivação que resultem na mesma forma sentencial: basta alterar a ordem em que os não-terminais são substituídos em cada passo da derivação.

4.4 Ambigüidade

Diz-se que uma gramática livre de contexto é **não-ambígua** se, para toda e qualquer cadeia pertencente à linguagem por ela gerada, existir uma única seqüência de derivações mais à esquerda e uma única seqüência de derivações mais à direita que a geram.

Diz-se que uma gramática livre de contexto é **ambígua** quando existir pelo menos uma cadeia, pertencente à linguagem por ela gerada, que possua mais de uma seqüência distinta de derivações, feitas exclusivamente através de substituições de não-terminais mais à esquerda ou mais à direita. Na verdade, a primeira condição implica a segunda e vice-versa, ou seja, se houver mais de uma seqüência com substituições mais à esquerda, então haverá também mais de uma com substituições mais à direita, conforme demonstrado no Teorema 4.2.

Teorema 4.2 (Derivações à esquerda ⇔ derivações à direita) *Se $w \in L(G)$, com G sendo uma gramática livre de contexto, e existindo duas (ou mais) derivações mais à esquerda para w em G, então existem também, correspondentemente, duas (ou mais) derivações mais à direita para w em G.*

Justificativa As gramáticas livres de contexto possuem a propriedade de que as derivações de cada um dos símbolos não-terminais presentes em uma mesma forma sentencial podem ser feitas de forma independente umas das outras. Se $S \Rightarrow^* \alpha X \beta \Rightarrow^* \alpha\gamma\beta$, então a derivação $X \Rightarrow \gamma$ independe de α e β.

Considere-se uma gramática livre de contexto G e a seqüência de derivações mais à esquerda $S \Rightarrow^* \alpha_1 X \gamma$, com $\alpha_1 \in \Sigma^*, X \in N$, e $\gamma \in V^*$. Considere-se que $X \Rightarrow^* \alpha_2$, com $\alpha_2 \in \Sigma^*$ e $\gamma \Rightarrow^* \alpha_3$, com $\alpha_3 \in \Sigma^*$. Então, $S \Rightarrow^* \alpha_1 X \gamma \Rightarrow^* \alpha_1\alpha_2\alpha_3$ e $\alpha_1\alpha_2\alpha_3 \in L(G)$.

Sejam $X \rightarrow \beta_1$ e $X \rightarrow \beta_2$ duas produções distintas de G. Então, existem duas derivações à esquerda distintas para a cadeia $\alpha_1\alpha_2\alpha_3$:

- $S \Rightarrow^* \alpha_1 X \gamma \Rightarrow \alpha_1\beta_1\gamma \Rightarrow^* \alpha_1\alpha_2\gamma \Rightarrow^* \alpha_1\alpha_2\alpha_3$
- $S \Rightarrow^* \alpha_1 X \gamma \Rightarrow \alpha_1\beta_2\gamma \Rightarrow^* \alpha_1\alpha_2\gamma \Rightarrow^* \alpha_1\alpha_2\alpha_3$

Se $\alpha_1\alpha_2\alpha_3 \in L(G)$ e $X \Rightarrow^* \alpha_2$, então existe uma seqüência de derivações mais à direita $S \Rightarrow^* \mu X \alpha_3$, com $\mu \in V^*$ e, além disso, $\mu \Rightarrow^* \alpha_1$. Portanto, $S \Rightarrow^* \mu X \alpha_3 \Rightarrow^* \mu \alpha_2\alpha_3 \Rightarrow^* \alpha_1\alpha_2\alpha_3$.

Como, por hipótese, $X \Rightarrow \beta_1 \Rightarrow^* \alpha_2$ e $X \Rightarrow \beta_2 \Rightarrow^* \alpha_2$, então existem duas seqüências de derivações mais à direita para a cadeia $\alpha_1\alpha_2\alpha_3$:

- $S \Rightarrow^* \mu X \alpha_3 \Rightarrow \mu \beta_1 \alpha_3 \Rightarrow^* \mu \alpha_2\alpha_3 \Rightarrow^* \alpha_1\alpha_2\alpha_3$
- $S \Rightarrow^* \mu X \alpha_3 \Rightarrow \mu \beta_2 \alpha_3 \Rightarrow^* \mu \alpha_2\alpha_3 \Rightarrow^* \alpha_1\alpha_2\alpha_3$

De maneira análoga, é possível demonstrar que, para cada seqüência distinta de derivações mais à esquerda que geram uma mesma cadeia da linguagem, existe uma seqüência distinta de derivações mais à direita correspondente, que gera a mesma cadeia■

Exemplo 4.12 A gramática cujo conjunto de regras está abaixo apresentado é equivalente à gramática anteriormente utilizada na definição da linguagem das expressões aritméticas sobre $\{a, +, *, (,)\}$ do Exemplo 4.1. Diferentemente daquela, no entanto, apenas um símbolo não-terminal (E) é utilizado:

$$\begin{array}{rcl} \{E & \rightarrow & E + E, \\ E & \rightarrow & E * E, \\ E & \rightarrow & a, \\ E & \rightarrow & (E)\} \end{array}$$

Adotando-se o critério das derivações mais à esquerda, pode-se perceber que a sentença $a+a*a$ pode ser derivada ao menos de duas formas distintas:

1. Aplicando-se inicialmente a produção $E \rightarrow E + E$:
   $E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$

2. Aplicando-se inicialmente a produção $E \rightarrow E * E$:
   $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$

Como conseqüência, a gramática acima é ambígua. É possível provar, no entanto, que a gramática do Exemplo 4.1 (com os não-terminais E, T e F) admite uma única seqüência de derivação para cada sentença pertencente à linguagem. Note-se, em particular, que isso ocorre com a sentença acima considerada. Como conclusão, esta linguagem pode ser indistintamente definida através de uma gramática ambígua ou de uma gramática não-ambígua. □

Exemplo 4.13
Considere-se o fragmento de gramática abaixo apresentado, que ilustra um problema típico de determinadas linguagens de programação de alto nível — a ambigüidade na construção de comandos condicionais aninhados.

<*programa*> → ... <*comando*> ...
<*comando*> → <*condicional*>
<*condicional*> → `if` <*condição*> `then` <*comando*>
<*condicional*> → `if` <*condição*> `else` <*comando*> `else` <*comando*>
<*condição*> → ...

Passa-se a analisar a estrutura da seguinte forma sentencial:

`if` <*condição*> `then if` <*condição*> `then` <*comando*> `else` <*comando*>

Para se determinar uma eventual ambigüidade nessa forma sentencial, escolhe-se inicialmente e fixa-se um critério de substituição de não-terminais — por exemplo, as substituições dos não-terminais mais à esquerda. Neste caso, é fácil verificar que a forma sentencial acima admite duas seqüências distintas de derivação:

1. Primeira alternativa:

<*programa*> ⇒ ... <*comando*> ...
⇒ ... <*condicional*> ...
⇒ ... `if` <*condição*> `then` <*comando*> `else` <*comando*> ...
⇒ ... `if` <*condição*> `then` <*condicional*> `else` <*comando*> ...
⇒ ... `if` <*condição*> `then if` <*condição*> `then` <*comando*>
`else` <*comando*> ...

2. Segunda alternativa:

<*programa*> ⇒ ... <*comando*> ...
⇒ ... <*condicional*> ...
⇒ ... `if` <*condição*> `then` <*comando*> ...
⇒ ... `if` <*condição*> `then` <*condicional*> ...
⇒ ... `if` <*condição*> `then if` <*condição*> `then` <*comando*>
`else` <*comando*> ...

No primeiro caso, o ramo do "`else`" é considerado como parte integrante do comando condicional mais externo e, no segundo, este mesmo ramo é considerado como parte do comando condicional mais interno:

`if` <*condição*> `then` <u>`if` <*condição*> `then` <*comando*></u> `else` <*comando*>

`if` <*condição*> `then` $\underbrace{\texttt{if } \langle\textit{condição}\rangle \texttt{ then } \langle\textit{comando}\rangle \texttt{ else } \langle\textit{comando}\rangle}$

Assim, a forma sentencial apresentada é ambígua, o que também torna ambígua a gramática. Como se pode verificar facilmente, as mesmas conclusões poderiam também ter sido obtidas caso fossem consideradas apenas substituições mais à direita. □

Na prática, apenas uma interpretação deve ser atribuída para comandos como o apresentado acima em cada possível implementação desta linguagem. No entanto, o que interessa é que a formalização gramatical dessa construção é ambígua, ou seja, ela admite mais de uma interpretação para a sentença em questão, independentemente da implementação considerada.

É certo que o termo "interpretação" denota significado, conteúdo semântico, assuntos que não estão sendo considerados no momento. Mas, como se sabe, o significado de uma sentença costuma estar muito associado à sua estrutura, e por esse motivo a existência de duas ou mais seqüências distintas de derivações para uma mesma sentença sugere a eventual existência de uma ambigüidade semântica a partir de uma ambigüidade sintática (ou estrutural) concreta constatada.

Ambigüidades sintáticas (simplesmente ambigüidades, neste texto) constituem geralmente uma característica indesejável das gramáticas de linguagens de programação e das demais linguagens artificiais em geral, uma vez que é comum, em casos como este, que a construção ambígua seja implementada (isto é, seja associada a interpretações semânticas) de formas diversas, dependendo da opção efetuada pelo projetista de cada compilador e/ou interpretador, prejudicando, assim, a portabilidade dos programas de aplicação entre uma implementação e outra.

Normalmente, ambigüidades costumam ser eliminadas através da utilização de construções gramaticais que impeçam a existência de mais de uma seqüência de derivações mais à esquerda (e, portant,o também mais à direita) para cada sentença pertencente à linguagem, desde que fixado e observado *a priori*, naturalmente, um critério único para a substituição dos não-terminais nas formas sentenciais. Eventualmente, como no Exemplo 4.14 apresentado a seguir, modificações mínimas podem vir a ser introduzidas na linguagem em conseqüência de tais alterações.

Assim, por exemplo, é comum, em linguagens de programação modernas, a definição de comandos condicionais de maneira semelhante ao abaixo apresentado.

Exemplo 4.14
Seja a seguinte gramática, inspirada na do Exemplo 4.13:

<*comando*>	→	... <*comando*> ...
<*comando*>	→	<*condicional*>
<*condicional*>	→	`if` <*condição*> `then` <*comando*> `fi`
<*condicional*>	→	`if` <*condição*> `then` <*comando*> `else` <*comando*> `fi`
<*condição*>	→	...

A associação do "`else`" ao comando condicional mais externo ou mais interno é feita, de acordo com esta nova gramática, respectivamente:

`if` <*condição*> `then` $\underbrace{\texttt{if } \langle\textit{condição}\rangle \texttt{ then } \langle\textit{comando}\rangle \texttt{ fi}}$ `else` <*comando*> `fi`

ou

`if` <*condição*> `then` $\underbrace{\texttt{if } \langle\textit{condição}\rangle \texttt{ then } \langle\textit{comando}\rangle \texttt{ else } \langle\textit{comando}\rangle \texttt{ fi}}$ `fi`

□

Note-se que, neste caso, sentenças equivalentes à anteriormente apresentada possuem apenas uma seqüência de derivações mais à esquerda ou à direita, sendo a gramática, portanto, isenta de ambigüidade.

Normalmente, o termo ambigüidade costuma ser relacionado apenas a sentenças e gramáticas da forma como foi mencionado anteriormente. Diz-se que uma linguagem livre de contexto é **ambígua** se ela puder ser gerada por uma gramática livre de contexto ambígua, e **não-ambígua** se ela puder ser gerada por uma gramática livre de contexto não-ambígua. No entanto, raramente se costuma dizer que uma linguagem é ambígua, uma vez que uma mesma linguagem pode ser gerada por inúmeras gramáticas distintas (ambíguas e não-ambíguas).

Há, no entanto, casos em que se pode provar, apesar de isso ser, em geral, uma tarefa consideravelmente complexa, que determinadas linguagens são **inerentemente ambíguas**, indicando com isso que toda e qualquer gramática que gera a linguagem em questão deve ser necessariamente ambígua. Em casos como esse, o conceito de ambigüidade pode ser estendido também para caracterizar mais um atributo da linguagem.

Exemplo 4.15
A linguagem $\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$ é inerentemente ambígua. A demonstração pode ser encontrada em [46]. □

Exemplo 4.16
A linguagem $\{a^i b^j c^k \mid i = j \text{ ou } j = k\}$ é inerentemente ambígua. A demonstração pode ser encontrada em [49]. □

Quando se consideram gramáticas livres de contexto, é natural que se questione a relação que existe entre o número de derivações canônicas que podem ser efetuadas para uma certa cadeia de uma linguagem e o número de árvores de derivação distintas que podem ser construídas para representar essa mesma cadeia. Este assunto será tratado no Teorema 4.3.

Teorema 4.3 (Múltiplas derivações ⇔ múltiplas árvores) *Seja G uma gramática livre de contexto. Para toda cadeia $w \in L(G)$, o número de seqüências distintas de derivações mais à esquerda (e portanto mais à direita) é igual ao número de árvores de derivação distintas que representam w.*

Justificativa Pode ser encontrada em [46]. Não é difícil, no entanto, intuir que, para cada forma sentencial obtida através de derivações mais à esquerda, a substituição do não-terminal mais à esquerda por produções distintas, sejam elas quantas forem, dá origem a uma quantidade idêntica de árvores distintas, as quais são diferenciadas pelos nós que representam os filhos do nó que representa esse não-terminal. De forma análoga, o raciocínio inverso também se aplica. ■

Por outro lado, quando se consideram gramáticas livres de contexto não-ambíguas, a ordem em que são feitas as substituições dos símbolos não-terminais nas formas sentenciais é irrelevante do ponto de vista das cadeias que podem ser geradas e das respectivas árvores de derivação. Como mostra o Teorema 4.4, qualquer que seja a ordem de substituição dos não-terminais escolhida na geração de uma mesma cadeia, a árvore de derivação será sempre a mesma.

Teorema 4.4 (Múltiplas derivações ⇔ única árvore) *Seja G uma gramática livre de contexto não-ambígua. Para toda cadeia $w \in L(G)$, toda e qualquer seqüência de derivações que produz w é representada através da mesma e única árvore de derivação.*

Justificativa Se G é uma gramática livre de contexto não-ambígua, então, conforme a definição, toda e qualquer cadeia $w \in L(G)$ possui uma única seqüência de derivações mais à esquerda ou mais à direita, mesmo que existam outras seqüências não-canônicas de derivação para essa mesma cadeia. Conforme o Teorema 4.3, existe uma única árvore de derivação para w. Logo, todas as derivações de w em G são representadas por uma mesma árvore de derivação. ■

Conforme a definição, uma gramática livre de contexto não-ambígua é aquela para a qual cada cadeia pertencente à linguagem gerada por tal gramática é gerada por uma única seqüência de derivações mais à esquerda e por uma única seqüência de derivações mais à direita. O Teorema 4.4 estende esse resultado, e estabelece que tais seqüências de derivações correspondem a uma mesma árvore de derivação.

Em resumo: cada uma das cadeias pertencentes a uma linguagem gerada por uma gramática livre de contexto não-ambígua possui uma única seqüência de derivações mais à esquerda, uma única seqüência de derivações mais à direita e uma única árvore de derivação.

Esta importante propriedade das linguagens livres de contexto (e portanto também das linguagens regulares) permite que se adote uma ordem arbitrária para a aplicação das regras gramaticais na derivação de uma sentença. Tecnologicamente, isso pode ser aproveitado escolhendo-se uma ordem que seja econômica do ponto de vista algorítmico.

A existência de uma única seqüência canônica de derivações (uma mais à esquerda e outra mais à direita) para cada sentença de uma linguagem gerada por uma gramática não-ambígua é explorada nos algoritmos de construção de reconhecedores sintáticos determinísticos para linguagens livres de contexto, cuja operação consiste exatamente na busca sistemática de tal seqüência (e, conseqüentemente, da correspondente árvore de derivação), se essa seqüência existir, para cada cadeia de entrada que lhe seja submetida. Do ponto de vista tecnológico, esse resultado permite a busca da solução de implementação mais barata, conforme a linguagem considerada.

Por outro lado, o Teorema 4.3 permite estender o conceito de gramática livre de contexto ambígua, que passa a ser caracterizado de três formas distintas, porém equivalentes entre si. Diz-se que uma gramática livre de contexto é ambígua se for possível identificar, na linguagem por ela gerada, pelo menos uma cadeia que:

- possa ser derivada por duas ou mais seqüências distintas de derivações mais à esquerda, ou
- possa ser derivada por duas ou mais seqüências distintas de derivações mais à direita, ou
- possa ser representada por duas ou mais árvores de derivação distintas.

Exemplo 4.17 Considere-se o fragmento de gramática utilizado para ilustrar o problema da ambigüidade em comandos condicionais (Exemplo 4.13). De acordo com essa gramática, a forma sentencial:

`if` <*condição*> `then if` <*condição*> `then` <*comando*> `else` <*comando*>

pode ser derivada segundo duas seqüências distintas de derivações mais à esquerda. Como conseqüência, existem também duas árvores de derivação distintas que possuem como fronteira essa mesma forma sentencial:

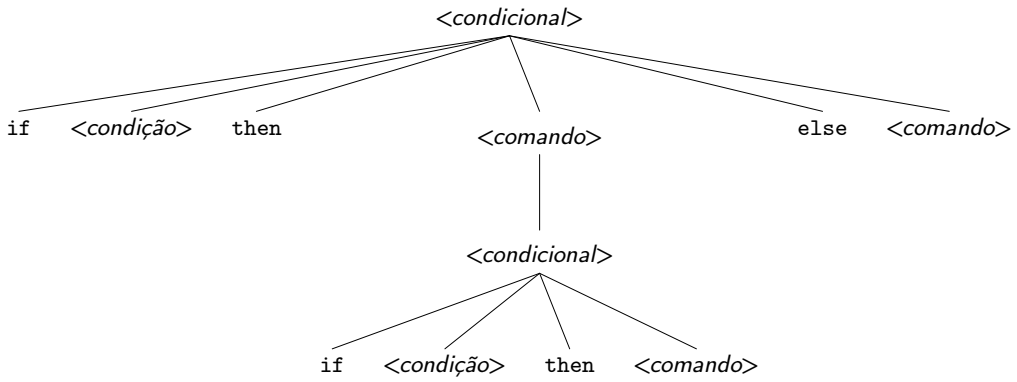

Figura 4.3: Ambigüidade no comando IF: associação interna

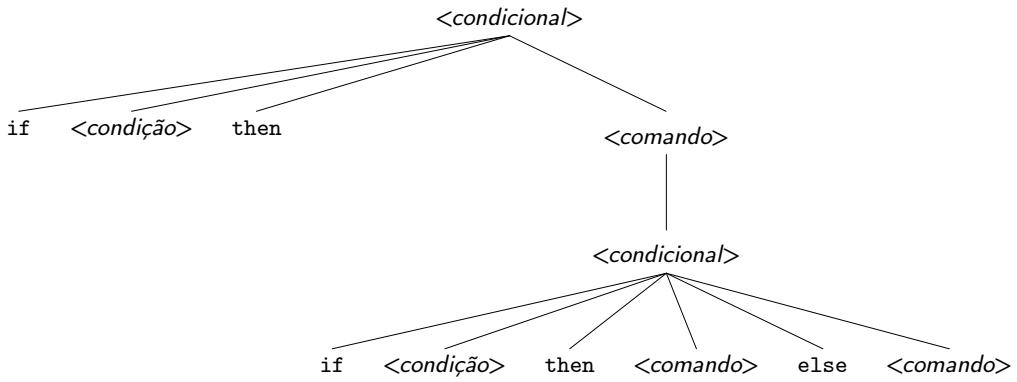

Figura 4.4: Ambigüidade no comando IF: associação externa

□

4.5 Simplificação de Gramáticas Livres de Contexto

Por uma questão de conveniência no estudo das gramáticas livres de contexto, muitas vezes é necessário submetê-las a simplificações que visam torná-las mais apropriadas para a demonstração de teoremas, para a verificação de propriedades, ou simplesmente para facilitar a sua análise.

Tais simplificações consistem em manipulações gramaticais que não afetam a linguagem definida pela gramática original, e são de três tipos (os novos termos serão definidos mais adiante):

1. Eliminação de símbolos inacessíveis e inúteis;
2. Eliminação de produções em vazio;
3. Eliminação de produções unitárias.

Para cada tipo de transformação será apresentado um algoritmo que mapeia a gramática original em uma outra gramática que exiba uma das propriedades acima enumeradas. Em todos os casos é possível demonstrar formalmente (ver [46]) que a transformação produz o efeito proposto sem alterar a linguagem definida pela gramática original.

Considere-se inicialmente a definição e eliminação de símbolos inacessíveis e inúteis. Dada uma gramática livre de contexto $G = (V, \Sigma, P, S)$, então:

- Diz-se que um símbolo $Y \in V$ — terminal ou não-terminal — é **inacessível**, se não for possível derivar qualquer forma sentencial em que se registre alguma ocorrência de Y. Caso contrário, o símbolo é dito **acessível**. Formalmente, símbolos acessíveis Y são aqueles para os quais existem derivações da forma $S \Rightarrow^* \alpha Y \beta$, com $\alpha, \beta \in V^*$. Símbolos inacessíveis, portanto, são aqueles para os quais inexistem derivações com tais características.

- Diz-se que um símbolo Y — não-terminal — é **inútil**, se não for possível derivar pelo menos uma cadeia formada exclusivamente por terminais (ou a cadeia vazia) a partir de Y. Caso contrário, o símbolo é dito **útil**. Formalmente, símbolos úteis Y são aqueles para os quais existem derivações da forma $Y \Rightarrow^* \gamma$, com $\gamma \in \Sigma^*$. Símbolos inúteis, portanto, são aqueles para os quais inexistem derivações com tais características. Cumpre notar que símbolos terminais são, por definição, sempre úteis.

Um símbolo Y que seja simultaneamente acessível e útil é, portanto, aquele que satisfaz à condição:

$$S \Rightarrow^* \alpha Y \beta \Rightarrow^* \alpha\gamma\beta, \quad \text{com } \alpha, \beta \in V^*, \gamma \in \Sigma^*$$

Em outras palavras, um símbolo Y é acessível e útil se e somente se:

1. Y está presente em pelo menos uma forma sentencial derivada a partir da raiz da gramática, e

2. Y deriva pelo menos uma cadeia pertencente ao conjunto Σ^* (condição verificada trivialmente pelos símbolos terminais).

A acessibilidade de um símbolo Y garante a existência de pelo menos uma forma sentencial derivável a partir da raiz da gramática em que Y tome parte. A utilidade garante que o símbolo Y gera pelo menos uma cadeia formada exclusivamente por símbolos terminais (ou a cadeia vazia).

Símbolos inacessíveis e inúteis em nada contribuem para a linguagem definida pela gramática livre de contexto em questão. Ainda que o cojunto de cadeias gerado por um símbolo inacessível seja diferente de vazio, a sua inacessibilidade torna-o irrelevante do ponto da vista da linguagem definida pela raiz da gramática. Por outro lado, mesmo se acessíveis, os símbolos inúteis sempre anulam a linguagem gerada por toda e qualquer forma sentencial a que possam pertencer.

Dessa forma, a idéia de se eliminarem símbolos inacessíveis e inúteis apresenta grande interesse, uma vez que permite obter gramáticas mais compactas, normalmente mais legíveis, e nas quais todo e qualquer símbolo contribui efetivamente com a linguagem gerada. Além disso, a inexistência de símbolos inacessíveis e inúteis é pré-requisito para a obtenção da algumas importantes formas normais.

A transformação de uma gramática em outra equivalente, isenta de símbolos inacessíveis e inúteis, pode ser feita em dois passos, através de dois algoritmos distintos: o primeiro algoritmo encarrega-se de eliminar da gramática original os símbolos inúteis; o segundo algoritmo elimina todos os símbolos inacessíveis.

Teorema 4.5 (Eliminação de símbolos inacessíveis e inúteis) *Toda linguagem livre de contexto pode ser gerada por uma gramática livre de contexto em que não há símbolos inacessíveis ou inúteis.*

Justificativa A demonstração deste teorema pode ser realizada tomando-se como base os dois algoritmos apresentados a seguir. O Algoritmo 4.1 mostra como eliminar os não-terminais que não geram cadeias de terminais, bem como as produções em que os mesmos comparecem, preservando em N apenas os símbolos do conjunto $N' = \{ Y \in N \mid Y \Rightarrow^* \alpha, \text{ com } \alpha \in \Sigma^* \}$.

Algoritmo 4.1 (Eliminação de símbolos inúteis) *Eliminação de símbolos inúteis em gramáticas livres de contexto.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$, tal que $L(G) \neq \varnothing$.
- Saída: uma gramática livre de contexto $G' = (V', \Sigma, P', S)$, tal que $L(G') = L(G)$ e $Y \in N'$ se e apenas se $L(Y) \neq \varnothing$.
- Método:

  1. $N_0 \leftarrow \varnothing$;
  2. $i \leftarrow 1$;
  3. $N_i \leftarrow N_{i-1} \cup \{ Y \mid Y \rightarrow \alpha \in P \text{ e } \alpha \in (N_{i-1} \cup \Sigma^*) \}$;
  4. Se $N_i \neq N_{i-1}$, então:

     a) $i \leftarrow i + 1$;

     b) Desviar para (3);

     Caso contrário:

     a) $N' \leftarrow N_i$;

     b) $P' \leftarrow \{ A \rightarrow X_1 X_2 \ldots X_n \in P \mid A, X_1, X_2, \ldots, X_n \in (N_i \cup \Sigma) \}$.

Observe-se que $N' \subseteq N$ e $P' \subseteq P$. Intuitivamente, este algoritmo funciona incluindo-se seletivamente em um novo conjunto de símbolos não-terminais (inicialmente vazio) apenas aqueles não-terminais que:

1. Derivam diretamente cadeias de símbolos terminais (primeira iteração do passo 2), ou

2. Derivam diretamente cadeias que incluem símbolos não-terminais, os quais, por sua vez, derivam cadeias de símbolos terminais, direta ou indiretamente.

Além disso, são preservadas em P' apenas as produções de P que fazem referências exclusivas a elementos de Σ e de N', pois as demais mostram-se sem utilidade. Note-se que o Algoritmo 4.1 pode ser usado para se determinar se $L(G) \neq \varnothing$ para uma dada gramática livre de contexto G — basta determinar, ao fim de sua execução, se $S \in N'$. Em caso afirmativo, $L(G) \neq \varnothing$; caso contrário, $L(G) = \varnothing$.

O segundo algoritmo mostra como eliminar os símbolos terminais e não-terminais que não fazem parte de qualquer forma sentencial derivável a partir da raiz da gramática, ou seja, $Y \in V'$ se e apenas se $\exists\, S \Rightarrow^* \alpha Y \beta,$ com $\alpha, \beta \in V^*$.

Algoritmo 4.2 (Eliminação de símbolos inacessíveis) *Eliminação de símbolos inacessíveis em gramáticas livres de contexto.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$.
- Saída: uma gramática livre de contexto $G' = (V', \Sigma', P', S)$, tal que $L(G') = L(G)$ e $Y \in V'$ se e apenas se $\exists\, S \Rightarrow^* \alpha Y \beta,$ com $\alpha, \beta \in V^*$.
- Método:

  1. $V_0 \leftarrow \{S\}$;
  2. $i \leftarrow 1$;
  3. $V_i \leftarrow V_{i-1} \cup \{X_j \in V, 1 \leq j \leq n \mid A \rightarrow X_1 X_2 \ldots X_n \in P \text{ e } A \in N_{i-1}\}$;
  4. Se $V_i \neq V_{i-1}$, então:

     a) $i \leftarrow i + 1$;

     b) Desviar para (3);

     Caso contrário:

     a) $N' \leftarrow V_i \cap N$;

     b) $\Sigma' \leftarrow V_i \cap \Sigma$;

     c) $P' \leftarrow \{A \rightarrow X_1 X_2 \ldots X_n \in P \mid A, X_1, X_2 \ldots X_n \in V_i\}$.

■

Observe-se a similaridade entre as definições e os algoritmos da presente seção com os da Seção 3.3, sobre autômatos finitos.

De fato, há uma correspondência direta entre os conceitos de símbolo e estado acessível, bem como entre os de símbolo e estado útil. Há, também, uma relação direta entre os algoritmos para a eliminação de símbolos inúteis (Algoritmo 4.1) e a para eliminação de símbolos inacessíveis (Algoritmo 4.2) com os algoritmos para a eliminação dos estados inúteis e inacessíveis em autômatos finitos (respectivamente, Algoritmos 3.11 e 3.9).

Trata-se, como se pode perceber, essencialmente dos mesmos conceitos e dos mesmos métodos.

No caso do Algoritmo 4.1, trata-se da identificação sistemática dos símbolos que geram alguma cadeia formada apenas por terminais. No Algoritmo 3.9, trata-se da identificação dos estados que conduzem a algum estado final do autômato. O Algoritmo 4.2 identifica os símbolos que podem ser gerados a partir da raiz S. O Algoritmo 3.11 identifica os estados que podem ser alcançados a partir do estado inicial do autômato.

Note-se também que os Algoritmos 4.1 e 4.2 não apenas eliminam símbolos dos conjuntos N e Σ da gramática original, no caso geral, como também eliminam todas as produções de P que envolvam quaisquer símbolos inacessíveis ou inúteis eliminados.

Símbolos inacessíveis e inúteis podem ser eliminados de uma gramática livre de contexto qualquer aplicando-se-lhes inicialmente o Algoritmo 4.1 e, ao resultado deste, o Algoritmo 4.2, ou vice-versa. A ordem de aplicação dos algoritmos não interfere no resultado final obtido, como demonstra o Teorema 4.6.

Teorema 4.6 (Eliminação de símbolos inacessíveis e inúteis) *A obtenção de uma gramática livre de contexto G_3, isenta de símbolos inacessíveis ou inúteis, a partir de uma gramática livre de contexto G_1 qualquer, pode ser feita pela aplicação dos Algoritmos 4.1 e 4.2 uma única vez cada, em qualquer seqüência.*

Justificativa Seja $G_1 = (V_1, \Sigma_1, P_1, S_1)$. Os símbolos do conjunto V_1 podem ser, individualmente, classificados em uma das seguintes categorias:

i) Acessível e útil;

ii) Acessível e inútil;

iii) Inacessível e útil;

iv) Inacessível e inútil.

Além disso, os Algoritmos 4.1 e 4.2 não criam novos símbolos, tampouco alteram as características de acessibilidade ou de utilidade dos símbolos da gramática de entrada que foram preservados na gramática de saída. Em função dessas observações, considerem-se as duas seguintes seqüências possíveis para a aplicação dos Algoritmos 4.1 e 4.2:

1. Inicialmente o Algoritmo 4.1, e depois o Algoritmo 4.2:

   A aplicação do Algoritmo 4.1 (eliminação de símbolos inúteis) à gramática G_1 resulta na gramática $G_2 = (V_2, \Sigma_2, P_2, S_2)$, em que os símbolos de V_2 são apenas dos tipos (i) e (iii). Em seguida, a aplicação do Algoritmo 4.2 (eliminação de símbolos inacessíveis) à gramática G_2 resulta na gramática $G_3 = (V_3, \Sigma_3, P_3, S_3)$, em que os símbolos de V_3 são apenas do tipo (i).

2. Inicialmente o Algoritmo 4.2, e depois o Algoritmo 4.1:

   A aplicação do Algoritmo 4.2 (eliminação de símbolos inacessíveis) à gramática G_1 resulta na gramática $G_2 = (V_2, \Sigma_2, P_2, S_2)$, em que os símbolos de V_2 são apenas dos tipos (i) e (ii). Em seguida, a aplicação do Algoritmo 4.1 (eliminação de símbolos inúteis) à gramática G_2 resulta na gramática $G_3 = (V_3, \Sigma_3, P_3, S_3)$, em que os símbolos de V_3 são apenas do tipo (i).

Logo, a ordem de aplicação dos algoritmos de eliminação de símbolos inacessíveis e inúteis é irrelevante, e a obtenção de uma gramática livre de contexto, isenta de tais símbolos, pode ser feita pela aplicação de cada um dos correspondentes algoritmos uma única vez, em qualquer ordem. ■

Exemplo 4.18 Considere-se $G = (V, \Sigma, P, S)$, com:

$$\begin{aligned} V &= \{S, A, B, C, a, b\} \\ \Sigma &= \{a, b\} \\ P &= \{S \to A \mid B, A \to aB \mid bS \mid b, B \to AB \mid Ba, C \to AS \mid b\} \end{aligned}$$

Aplicando-se o algoritmo de eliminação de não-terminais que não geram cadeias de terminais, obtém-se:

$$\begin{aligned} N_0 &= \varnothing \\ N_1 &= \{A, C\} \\ N_2 &= \{A, C, S\} \\ N_3 &= \{A, C, S\} \end{aligned}$$

Logo, $G' = \{V', \Sigma, P', S\}$, com:

$$\begin{aligned} V' &= \{S, A, C, a, b\} \\ P' &= \{S \to A \mid A \to bS \mid b, C \to AS \mid b\} \end{aligned}$$

Note-se que o não-terminal B foi eliminado de N uma vez que não é possível derivar qualquer cadeia de terminais a partir dele. Aplicando-se a G' o Algoritmo 4.2, para a eliminação de símbolos inacessíveis, obtém-se:

$$\begin{aligned} V_0 &= \{S\} \\ V_1 &= \{S, A\} \\ V_2 &= \{S, A, b\} \end{aligned}$$

e $G'' = \{V'', \Sigma'', P'', S\}$, com:

$$\begin{aligned} V'' &= \{S, A\} \\ \Sigma'' &= \{b\} \\ P'' &= \{S \to A \mid A \to bS \mid b\} \end{aligned}$$

Observe-se a eliminação dos símbolos C e a de V'', uma vez que eles não comparecem em qualquer forma sentencial derivável a partir de S. G'' corresponde, assim, a uma gramática equivalente a G, isenta de símbolos inacessíveis e inúteis, e $L(G) = L(G'') = b^+$. □

Analisa-se agora a questão da eliminação das produções em vazio. **Produções em vazio** são produções da forma $A \to \epsilon$, e a total eliminação de produções desse tipo de uma gramática G naturalmente só é possível se $\epsilon \notin L(G)$.

Teorema 4.7 (Eliminação de produções em vazio) *Toda linguagem livre de contexto que não contém a cadeia vazia pode ser gerada por uma gramática livre de contexto em que não há produções em vazio.*

Justificativa O Algoritmo 4.3 mostra como transformar uma gramática G em outra equivalente G', isenta de produções em vazio. Se $\epsilon \in L(G)$, então será admitida uma única produção em vazio, cujo lado esquerdo corresponde à raiz da gramática, de modo que $L(G') = L(G)$.

Algoritmo 4.3 (Eliminação de produções em vazio) *Eliminação de produções em vazio em gramáticas livres de contexto.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$.
- Saída: uma gramática livre de contexto $G' = (N' \cup \Sigma, \Sigma, P', S')$, tal que $L(G') = L(G)$ e

  i) Se $\epsilon \notin L(G)$, então não há produções em vazio em G', ou

  ii) Se $\epsilon \in L(G)$, então a única produção em vazio em G' é $S' \rightarrow \epsilon$, onde S' é a raiz de G' e S' não aparece no lado direito de nenhuma produção.

- Método:

  1. $E_0 \leftarrow \{A \mid A \rightarrow \epsilon \in P\}$;
  2. $i \leftarrow 1$;
  3. $E_i \leftarrow E_{i-1} \cup \{A \mid A \rightarrow X_1 X_2 \ldots X_n \in P \text{ e } X_1, X_2 \ldots X_n \in E_{i-1}\}$;
  4. Se $E_i \neq E_{i-1}$, então:

     a) $i \leftarrow i + 1$;

     b) Desviar para (3);

     Caso contrário:

     a) $E \leftarrow E_{i-1}$;

  5. $P' \leftarrow \varnothing$;
  6. $P' \leftarrow \{A \rightarrow \beta \in P \mid \beta \neq \epsilon\}$;
  7. Considerem-se as produções de P' no formato:

     $A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \ldots B_k \alpha_k$, com $\alpha_i \in (V - E)^+$ e $B_i \in E$

     A versão final do conjunto P' é obtida acrescentando-se à sua versão anterior o conjunto das produções obtidas pela substituição dos símbolos B_i, $0 \leq i \leq k$ por ϵ, considerando-se todas as combinações possíveis, sem no entanto gerar a produção $A \rightarrow \epsilon$.

  8. Se $S \in E$, então:

     a) $P' \leftarrow P' \cup \{S' \rightarrow S \mid \epsilon\}$;

     b) $N' \leftarrow N \cup \{S'\}$;

Caso contrário:

a) $N' \leftarrow N$;

b) $S' \leftarrow S$.

Este algoritmo implementa um mapeamento do conjunto P no conjunto P', em que as seguintes condições são obedecidas:

1. Produções em vazio $A \rightarrow \epsilon \in P$ são eliminadas;

2. As produções $A \rightarrow \alpha \in P$, $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$, $\sigma_i \in V$, em que α contém símbolos não-terminais que eventualmente derivam ϵ, isto é, $\sigma_i \Rightarrow^* \epsilon$, $1 \leq i \leq n$, são expandidas para contemplar todos os possíveis casos de eliminação desses não-terminais, compensando assim a eliminação das produções em vazio.

Os passos (1), (2), (3) e (4) do algoritmo buscam identificar o subconjunto E dos elementos A de N, $E \subseteq N$, tais que $A \Rightarrow^* \epsilon$, e em muito se assemelham aos Algoritmos 4.1 e 4.2 anteriormente apresentados. Os passos (5), (6) e (7) realizam o mapeamento acima descrito, eliminando as produções em vazio e eventualmente expandindo as demais produções.

Observe-se que o fato de cada produção de P ser considerada no formato genérico:

$$A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \ldots B_k \alpha_k$$

visa isolar exatamente os não-terminais B_i, tais que $B_i \Rightarrow^* \epsilon$. A seguir são geradas, a partir de cada produção deste tipo, 2^k (no máximo) novas produções distintas, cada qual correspondente a uma particular combinação de não-terminais B_i substituídos por ϵ.

Por fim, note-se que se $S \in E$, isso indica que $\epsilon \in L(G)$. Assim, para que $L(G') = L(G)$, um novo símbolo não-terminal S' e a produção $S' \rightarrow \epsilon$ são introduzidos em G' no passo (8). ■

Exemplo 4.19 Seja uma gramática $G = (V, \Sigma, P, S)$, com:

$$\begin{array}{lcl} V & = & \{S, A, B, C\} \\ \Sigma & = & \{a, b\} \\ P & = & \{S \rightarrow ABC, A \rightarrow BB \mid \epsilon, B \rightarrow CC \mid a, C \rightarrow AA \mid b\} \end{array}$$

Como se pode perceber, $E = \{A, C, B, S\}$. Em conseqüência, obtém-se o seguinte conjunto de produções P':

$$\begin{array}{rcl} \{C & \rightarrow & AA \mid A \mid b, \\ B & \rightarrow & CC \mid C \mid a, \\ A & \rightarrow & BB \mid B, \\ S & \rightarrow & ABC \mid A \mid B \mid C \mid AB \mid AC \mid BC\} \end{array}$$

Pelo fato de $S \in E$, a produção $S' \rightarrow S \mid \epsilon$ deve ainda ser incorporada a P', resultando assim $G' = (V \cup \{S'\}, \Sigma, P', S')$. □

Exemplo 4.20 Considere-se a gramática $G = (V, \Sigma, P, S)$, com

$$\begin{aligned} V &= \{S, B, C\} \\ \Sigma &= \{a, b, c, d\} \\ P &= \{S \rightarrow aBC, B \rightarrow bB \mid \epsilon, C \rightarrow cCc \mid d \mid \epsilon\} \end{aligned}$$

Neste caso, $E = \{B, C\}$. Portanto, P' se torna:

$$\begin{aligned} \{B &\rightarrow bB \mid b, \\ C &\rightarrow cCc \mid cc \mid d, \\ S &\rightarrow aBC \mid aB \mid aC\} \end{aligned}$$

Como $\epsilon \notin L(G)$, então $G' = (V, \Sigma, P', S)$. □

O último caso de simplificação de gramáticas livres de contexto refere-se à eliminação de produções unitárias. **Produções unitárias** são produções da forma $A \rightarrow B$, em que A e B são não-terminais, e costumam ser descartadas das gramáticas livres de contexto porque nada acrescentam às formas sentenciais às quais são aplicadas, constituindo mera renomeação de símbolos (no caso, de A para B).

Teorema 4.8 (Eliminação de produções unitárias) *Toda linguagem livre de contexto pode ser gerada por uma gramática livre de contexto isenta de produções unitárias.*

Justificativa O Algoritmo 4.4 mostra como transformar gramáticas livres de contexto arbitrárias em outras equivalentes sem produções unitárias.

Algoritmo 4.4 (Eliminação de produções unitárias) *Eliminação de produções unitárias em gramáticas livres de contexto.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$.
- Saída: uma gramática livre de contexto $G' = (V, \Sigma, P', S)$, tal que $L(G') = L(G)$ e G' não contém produções unitárias.
- Método:

1. Para cada $A \in N$, constrói-se N_A tal que $N_A = \{B \in N \mid A \Rightarrow^* B$[2]$\}$ da seguinte forma:

   a) $N_0 \leftarrow \{A\}$;

   b) $i \leftarrow 1$;

   c) $N_i \leftarrow N_{i-1} \cup \{C \mid B \rightarrow C \in P \text{ e } B \in N_{i-1}\}$;

   d) Se $N_i \neq N_{i-1}$, então:

      i. $i \leftarrow i + 1$;

      ii. Desviar para (1.c);

[2]Devem-se considerar apenas as derivações que são obtidas pela aplicação exclusiva de regras unitárias.

Caso contrário:

i. $N_A \leftarrow N_{i-1}$;

2. $P' \leftarrow \{A \rightarrow \alpha \in P \mid \alpha \notin N\}$;

3. Para todo $B \in N_A$, se $B \rightarrow \alpha \in P$, e $\alpha \notin N$, então $P' \leftarrow P' \cup \{A \rightarrow \alpha\}$.

O funcionamento deste algoritmo baseia-se inicialmente na identificação e na associação de subconjuntos de N a cada não-terminal X da gramática, sendo que cada elemento Y desse subconjunto satisfaz à condição $X \Rightarrow^* Y$. Esses subconjuntos são construídos no passo (1) do algoritmo.

Observe-se que a condição $X \Rightarrow^* Y$ refere-se apenas a derivações efetuadas exclusivamente através do emprego de produções unitárias. Em alguns casos, é possível que tal condição seja satisfeita através do emprego de produções não-unitárias, isto é, $X \Rightarrow^* \alpha \Rightarrow^* Y, \mid \alpha \mid > 1$, como ilustrado no Exemplo 4.21.

Exemplo 4.21 Considere-se a gramática G:

$$\begin{array}{rcl} G & = & (\{S, X, Y, a\}, \{a\}, P, S) \\ P & = & \{S \rightarrow XY, \\ & & X \rightarrow a, \\ & & Y \rightarrow \epsilon\} \end{array}$$

A derivação $S \Rightarrow XY \Rightarrow X$, ou simplesmente $S \Rightarrow^* X$, é feita sem o uso de regras unitárias□

Para evitar situações como as do Exemplo 4.21, é suficiente garantir que a gramática em questão não possua regras em vazio, ou, alternativamente, desconsiderar derivações que façam uso das mesmas no cálculo dos conjuntos N_A (passo (1) do Algoritmo 4.4).

O conjunto P' é construído nos passos (2) e (3). Ele é obtido pela eliminação de todas as produções unitárias de P, preservando-se as demais e criando novas produções do tipo $X \Rightarrow^* \alpha$, onde $Y \in N_X$ e $Y \rightarrow \alpha \in P$. Isso implica "abreviar" uma seqüência de derivações, substituindo cada derivação que utilize produções unitárias pela derivação — utilizando as novas produções — que produz o efeito final pretendido. ■

Exemplo 4.22 Considere-se $G = (V, \Sigma, P, S)$, com o conjunto de regras P seguinte:

$$\begin{array}{rcl} \{S & \rightarrow & A \mid B \mid C, \\ A & \rightarrow & aaAa \mid B \mid \epsilon, \\ B & \rightarrow & bBb \mid b \mid C, \\ C & \rightarrow & cC \mid \epsilon\} \end{array}$$

Aplicando-se o Algoritmo 4.4, obtém-se:

$$\begin{array}{rcl} N_S & = & \{S, A, B, C\} \\ N_A & = & \{A, B, C\} \\ N_B & = & \{B, C\} \\ N_C & = & \{C\} \end{array}$$

Assim, $G' = (V, \Sigma, P', S)$, e P' torna-se:

$$\begin{aligned}\{S &\rightarrow aaAa \mid \epsilon \mid bBb \mid b \mid cC,\\ A &\rightarrow aaAa \mid \epsilon \mid bBb \mid b \mid cC,\\ B &\rightarrow bBb \mid b \mid cC \mid \epsilon,\\ C &\rightarrow cC \mid \epsilon\}\end{aligned}$$

A título de ilustração, considere-se a derivação da sentença $aabba$. Tomando-se como base a gramática G, a seqüência de derivações desta sentença é:

$$S \Rightarrow A \Rightarrow aaAa \Rightarrow aaBa \Rightarrow aabBba \Rightarrow aabCba \Rightarrow aabba$$

ao passo que, tomando-se como base a gramática modificada G', a seqüência de derivação desta sentença se torna:

$$S \Rightarrow aaAa \Rightarrow aabBba \Rightarrow aabba$$

(note-se como a eliminação de produções unitárias "abrevia" a derivação da sentença). □

Diz-se que uma gramática livre de contexto isenta de produções em vazio é **acíclica** se e somente se não existir $A \in N$, tal que $A \Rightarrow^* A$. Caso contrário, diz-se que a gramática é **cíclica**. O Algoritmo 4.4 pode ser facilmente adaptado para detectar ciclos em gramáticas livres de contexto isentas de produções em vazio, conforme apresentado no Algoritmo 4.5.

Algoritmo 4.5 (Detecção de ciclos) *Detecção de ciclos em gramáticas livres de contexto isentas de produções em vazio.*

- Entrada: uma gramática livre de contexto isenta de produções em vazio $G = (V, \Sigma, P, S)$.
- Saída: SIM, se G for cíclica. NÃO, se G for acíclica.
- Método:

  1. Para cada $A \in N$, constrói-se N_A tal que $N_A = \{B \in N \mid A \Rightarrow^* B\}$ da seguinte forma:

     a) $N_0 \leftarrow \{B \mid A \rightarrow B \in P \text{ e } B \neq A\}$;

     b) $i \leftarrow 1$;

     c) $N_i \leftarrow N_{i-1} \cup \{C \mid B \rightarrow C \in P \text{ e } B \in N_{i-1}\}$;

     d) Se $N_i \neq N_{i-1}$, então:

        i. $i \leftarrow i + 1$;

        ii. Desviar para (1.c);

     Caso contrário:

        i. $N_A \leftarrow N_{i-1}$;

  2. Se $\forall\, A \in N,\ A \notin N_A$, então NÃO; caso contrário, SIM.

4.6 Formas Normais para Gramáticas Livres de Contexto

Como se sabe, as gramáticas livres de contexto se caracterizam por apresentar produções p que seguem a forma geral $A \rightarrow \alpha$, $A \in N$, $\alpha \in V^*$. Conforme foi estudado na Seção 4.5, tais gramáticas podem sofrer simplificações — traduzidas, em alguns casos, em restrições impostas ao formato admitido para as produções $p \in P$ — que visam torná-las mais concisas, sem no entanto alterar a linguagem por elas definida.

Por outro lado, certos teoremas que serão apresentados no restante deste capítulo, como é, por exemplo, o caso daquele que estabelece a equivalência das gramáticas do tipo 2 com a classe de linguagens reconhecida pelos autômatos de pilha (Teorema 4.13), podem ser mais facilmente demonstrados considerando-se que as gramáticas livres de contexto estejam normalizadas em relação a algum padrão notacional.

Uma gramática é dita **normalizada** em relação a um certo padrão quando todas as suas produções seguem as restrições impostas pelo padrão em questão. É comum designar tais padrões como **formas normais**. Nesta seção serão definidas duas das formas normais mais importantes para as gramáticas livres de contexto: a Forma Normal de Chomsky e a Forma Normal de Greibach. Especial atenção será dedicada ao estudo de algoritmos que permitem o mapeamento de gramáticas do tipo 2 quaisquer em gramáticas equivalentes, aderentes a uma ou outra dessas formas normais. Mostrar-se-á que toda e qualquer gramática do tipo 2 corresponde a gramáticas equivalentes, expressas em ambas as formas normais.

Diz-se que uma gramática $G = (V, \Sigma, P, S)$ do tipo 2 obedece à **Forma Normal de Chomsky** ([62]) se todas as produções $p \in P$ forem de uma das duas formas seguintes:

1. $A \rightarrow BC$, ou
2. $A \rightarrow a$

com A, B, $C \in N$ e $a \in \Sigma$.

Se $\epsilon \in L(G)$, então admite-se $S \rightarrow \epsilon$ como única produção em que ϵ comparece do lado direito.

Teorema 4.9 (Forma Normal de Chomsky) *Toda linguagem livre de contexto L pode ser gerada por uma gramática livre de contexto na Forma Normal de Chomsky.*

Justificativa Seja G uma gramática livre de contexto qualquer que gera L. Sem perda de generalidade, considere-se que G não apresenta produções unitárias, símbolos inúteis e nem produções em vazio — exceto $S \rightarrow \epsilon$, se $\epsilon \in L(G)$. É possível demonstrar formalmente (ver [46]) que o mapeamento de G em G', a gramática equivalente na Forma Normal de Chomsky, pode ser efetuado através do Algoritmo 4.6.

Algoritmo 4.6 (Forma Normal de Chomsky) *Obtenção de uma gramática livre de contexto na Forma Normal de Chomsky.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$ isenta de produções unitárias, símbolos inúteis e produções em vazio.
- Saída: uma gramática livre de contexto $G' = (V', \Sigma, P', S)$, na Forma Normal de Chomsky, tal que $L(G) = L(G')$.
- Método:

1. $P' \leftarrow \varnothing$;
2. $N' \leftarrow N$;
3. Se $A \rightarrow BC \in P$, com $A, B, C \in N$, então $A \rightarrow BC \in P'$;
4. Se $A \rightarrow \sigma \in P$, com $A \in N, \sigma \in \Sigma$, então $A \rightarrow \sigma \in P'$;
5. Se $S \rightarrow \epsilon \in P$, então $S \rightarrow \epsilon \in P'$;
6. Para cada produção $p \in P$ da forma:

   $A \rightarrow X_1 X_2, \ldots, X_n$, com $n \geq 2$

   se $X_i \in \Sigma$, então criam-se novos não-terminais Y_i e produções $Y_i \rightarrow X_i$ substituindo-se as ocorrências de X_i por Y_i em p. Acrescentam-se os novos não-terminais Y_i a N' e as novas produções a P'.
7. Para cada produção da forma:

   $A \rightarrow X_1 X_2, \ldots, X_n$, com $n \geq 2$ e $X_i \in N$, $1 \leq i \leq n$

   gerada no passo (6), criar um novo conjunto de não-terminais Z_i e de produções da forma:

$$\begin{aligned} \{A &\rightarrow X_1 Z_1, \\ Z_1 &\rightarrow X_2 Z_2, \\ &\ldots \\ Z_{n-2} &\rightarrow X_{n-1} X_n\} \end{aligned}$$

   acrescentando-os, respectivamente, aos conjuntos N' e P'.

Trata-se, como se pode perceber, de um algoritmo bastante intuitivo. Inicialmente, todas as produções de P que originalmente já se encontram na Forma Normal de Chomsky são transferidas para P'. No passo (6), as produções com mais de um símbolo do lado direito são transformadas em produções contendo apenas símbolos não-terminais, devendo-se criar um novo símbolo não-terminal e uma nova produção para cada símbolo terminal a ser eliminado. Finalmente, no passo (7), cada produção neste formato é desmembrada em $n - 1$ novas produções, cada qual com apenas dois símbolos não-terminais do lado direito e, portanto, aderente à Forma Normal de Chomsky. ■

Exemplo 4.23 Considere-se a gramática $G = (V, \Sigma, P, S)$ com $N = \{E, T, F\}$ e o conjunto de produções P apresentado abaixo:

$$\begin{aligned} \{E &\rightarrow E + T \mid T, \\ T &\rightarrow T * F \mid F, \\ F &\rightarrow (E) \mid a\} \end{aligned}$$

Da aplicação do algoritmo acima resulta $G' = (V', \Sigma, P', S)$, com:

$$
\begin{aligned}
N' &= \{E, T, F, [+T], [+], [*F], [*], [(], [E)], [)]\} \\
P' &= \{E \to E[+T] \mid T, \\
&\quad [+T] \to [+]T, \\
&\quad [+] \to +, \\
&\quad T \to T[*F] \mid F, \\
&\quad [*F] \to [*]F, \\
&\quad [*] \to *, \\
&\quad F \to [(][E)] \mid a, \\
&\quad [(] \to (, \\
&\quad [E)] \to E[)]\}
\end{aligned}
$$

□

Diz-se que uma gramática livre de contexto $G = (V, \Sigma, P, S)$ obedece à **Forma Normal de Greibach** ([63]) se todas as suas produções $p \in P$ forem da forma:

$$A \to \sigma\alpha, \sigma \in \Sigma, \alpha \in N^*$$

Se $\epsilon \in L(G)$, então admite-se $S \to \epsilon$ como única produção em que ϵ comparece do lado direito. Como pré-requisito antes de apresentarmos o teorema que mostra como converter gramáticas livres de contexto quaisquer em equivalentes na Forma Normal de Greibach, será necessário apresentar um algoritmo que permita eliminar recursões à esquerda em gramáticas livres de contexto quaisquer.

Eliminar recursões à esquerda de uma gramática livre de contexto $G = (V, \Sigma, P, S)$ significa obter $G' = (V', \Sigma, P', S)$, de modo que $L(G) = L(G')$ e nenhum $A \in N'$ seja recursivo à esquerda. A eliminação de recursões à esquerda em gramáticas livres de contexto pode ser feita através do Algoritmo 4.7, cuja idéia básica consiste em transformar G, com não-terminais ordenados arbitrariamente, em G', de modo que $X_i \to X_{j\alpha}$ pertence a P' se e apenas se $j > i$.

Sem perda de generalidade, admite-se que a gramática G de entrada seja isenta de produções unitárias, símbolos inúteis e produções em vazio.

Algoritmo 4.7 (Recursões à esquerda) *Eliminação de recursões à esquerda em gramáticas livres de contexto.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$ isenta de produções unitárias, símbolos inúteis e produções em vazio.
- Saída: uma gramática livre de contexto $G' = (V', \Sigma, P', S)$ sem recursões à esquerda e tal que $L(G) = L(G')$.
- Método:

  1. $N' \leftarrow N$;
  2. $P' \leftarrow \varnothing$;
  3. $i \leftarrow 1$;
  4. Considerem-se os elementos de N ordenados segundo um critério arbitrário, $X_1, X_2, \ldots, X_p$;

5. Considerem-se todas as alternativas de substituição para X_i agrupadas na forma:

$$X_i \rightarrow X_i\alpha_1 \mid \ldots \mid X_i\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$$

em que todos os $\alpha_k, 1 \leq k \leq m$, não contenham X_i, e todos os β_k, $1 \leq k \leq n$, começam com símbolos diferentes de X_i. Se $\beta_k = X_j\gamma$, $j < i$, substituir X_j em $X_j\gamma$ pelas suas alternativas conforme as produções em que X_j comparece do lado esquerdo. Ou seja, se $X_j \rightarrow \delta_1 \mid \ldots \mid \delta_r$, fazer $\beta_k \rightarrow \delta_1\gamma \mid \ldots \mid \delta_r\gamma$.

Repetir este passo até que $\forall\, k, 1 \leq k \leq n,$ se $\beta_k = X_j\gamma,$ então $j \geq i$.

6. Neste passo, todas as alternativas de substituição para X_i podem ser reagrupadas na forma:

$$X_i \rightarrow X_i\alpha_1 \mid \ldots \mid X_i\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$$

em que β_i se inicia com $\sigma \in \Sigma$ ou $X_j \in N$, $j > i$. Introduzir um novo não-terminal X_i' e criar as produções abaixo em substituição às acima agrupadas:

$$\begin{array}{lcl} X_i & \rightarrow & \beta_1 \mid \ldots \mid \beta_n \mid \beta_1 X_i' \mid \ldots \mid \beta_n X_i' \\ X_i' & \rightarrow & \alpha_1 \mid \ldots \mid \alpha_m \mid \alpha_1 X_i' \mid \ldots \mid \alpha_m X_i' \end{array}$$

Fazer $N' \leftarrow N' \cup \{X_i'\}$ e acrescentar os dois conjuntos de produções acima a P'.

7. Se $i \neq p$:

   a) $i \leftarrow i + 1$;

   b) Desviar para (5).

Partindo de uma ordenação arbitrária dos não-terminais da gramática, o funcionamento deste algoritmo baseia-se na realização de substituições sucessivas de não-terminais de "ordem inferior" que comparecem como símbolo inicial de alternativas de substituição de não-terminais de "ordem superior" pelas suas respectivas definições. Note-se que, ao término de cada execução do passo (5), todas as alternativas de substituição de não-terminais X_i possuem necessariamente a forma:

$$X_i \rightarrow X_i\alpha_i \mid \ldots \mid X_i\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$$

O caso em que $i = 1$ é trivial, pois a condição acima é sempre imediatamente satisfeita. Se $i > 1$, e $\beta_k = X_j\gamma$, com $j < i$, então a substituição de X_j pelas suas respectivas produções, e assim sucessivamente, até um máximo de $i - j$ vezes, resulta sempre em $\beta_k = X_s\gamma$, $s \geq i$. Para facilitar a manipulação, o caso $s = i$ é representado separadamente dos demais casos em que $k > i$.

O passo (6) visa eliminar recursões diretas à esquerda nas produções de X_i. Assim, após a aplicação deste passo, todas as produções de X_i iniciam-se exclusivamente com símbolos terminais ou não-terminais de ordem $j > i$. Repetindo-se os passos (5) e (6) sucessivamente para $i = 1, 2, \ldots, p$, obtém-se G' isenta de recursões à esquerda. Note-se,

como conseqüência da aplicação deste algoritmo, que todas as produções de X_p se iniciam exclusivamente com símbolos terminais.

Para finalizar, observe-se que a introdução dos não-terminais X_i', e suas respectivas produções, no passo (6) não prejudica a intenção de eliminação das recursões à esquerda. Isso se deve aos seguintes motivos:

1. Como $\beta_j \neq \varnothing$, $1 \leq j \leq n$, pois não se admitem produções em vazio em G, X_i' nunca poderá comparecer como símbolo inicial de $\beta_j X_i'$ — alternativas de substituição de X_i —, prevenindo dessa maneira a eventualidade de introdução de recursões indiretas à esquerda.

2. Como $\alpha_j \neq \varnothing$, $1 \leq j \leq m$, pois por hipótese não se admitem produções unitárias em G, X_i' nunca poderá comparecer como símbolo inicial de $\alpha_j X_i'$ — alternativas de substituição de X_i' —, prevenindo dessa maneira a eventualidade de introdução de recursões diretas à direita.

Exemplo 4.24 Considere-se a gramática $G = (V, \Sigma, P, S)$ com $N = \{E, T, F\}$, sendo P o conjunto apresentado abaixo:

$$\begin{aligned} \{E &\rightarrow E + T \mid T, \\ T &\rightarrow T * F \mid F, \\ F &\rightarrow (E) \mid a\} \end{aligned}$$

Considere-se a ordenação $E < T < F$ para os não-terminais de G. A aplicação do Algoritmo 4.7 resulta na gramática G' com o seguinte conjunto de produções:

$$\begin{aligned} \{E &\rightarrow T \mid TE', \\ E' &\rightarrow +T \mid +TE', \\ T &\rightarrow F \mid FT', \\ T' &\rightarrow *F \mid *FT', \\ F &\rightarrow (E) \mid a\} \end{aligned}$$

□

Exemplo 4.25 Considere-se o conjunto abaixo de regras de uma gramática G, e a ordenação de seus não-terminais $X_1 < X_2 < X_3 < X_4$:

$$\begin{aligned} \{X_1 &\rightarrow X_2X_3 \mid a, \\ X_2 &\rightarrow X_3X_2 \mid X_2b \mid b, \\ X_3 &\rightarrow X_4X_1 \mid c, \\ X_4 &\rightarrow X_1X_3 \mid X_3e \mid d\} \end{aligned}$$

As produções de X_1 claramente satisfazem aos requisitos dos passos (5) e (6) do Algoritmo 4.7. As produções de X_2 satisfazem ao passo (5), porém não ao passo (6). Assim, elas são substituídas por:

$$\begin{aligned} X_2 &\rightarrow X_3X_2 \mid b \mid X_3X_2X_2' \mid bX_2' \\ X_2' &\rightarrow bX_2' \mid b \end{aligned}$$

Também as produções do não-terminal X_3 satisfazem às exigências dos passos (5) e (6), mas o mesmo não ocorre com as produções de X_4, mais especificamente com a produção $X_4 \rightarrow X_1X_3$,

que deve ser alterada conforme o determinado pelo passo (5). Substituindo-se X_1 pela sua definição, obtém-se:

$$X_4 \rightarrow X_2X_3X_3 \mid aX_3 \mid X_3e \mid d$$

Prosseguindo, substitui-se X_2 pela sua definição, obtendo-se:

$$X_4 \rightarrow X_3X_2X_3X_3 \mid bX_3X_3 \mid X_3X_2X_2'X_3X_3 \mid bX_2'X_3X_3 \mid aX_3 \mid X_3e \mid d$$

Finalmente, o passo (5) é completado substituindo-se X_3 pela sua definição, obtendo-se:

$$\begin{aligned} X_4 \rightarrow\ & X_4X_1X_2X_3X_3 \mid cX_2X_3X_3 \mid bX_3X_3 \mid X_4X_1X_2X_2'X_3X_3 \mid \\ & cX_2X_2'X_3X_3 \mid bX_2'X_3X_3 \mid aX_3 \mid X_4X_1e \mid ce \mid d \end{aligned}$$

Prosseguindo com a aplicação do passo (5) às produções de X_4, obtém-se:

$$\begin{aligned} X_4 \rightarrow\ & cX_2X_3X_3 \mid bX_3X_3 \mid cX_2X_2'X_3X_3 \mid bX_2'X_3X_3 \mid aX_3 \mid ce \mid d \mid cX_2X_3X_3X_4' \mid bX_3X_3X_4' \mid \\ & cX_2X_2'X_3X_3X_4' \mid bX_2'X_3X_3X_4' \mid aX_3X_4' \mid ceX_4' \mid dX_4' \\ X_4' \rightarrow\ & X_1X_2X_3X_3 \mid X_1X_2X_2'X_3X_3 \mid X_1e \mid X_1X_2X_3X_3X_4' \mid X_1X_2X_2'X_3X_3X_4' \mid X_1eX_4' \end{aligned}$$

Como resultado final da aplicação do algoritmo de eliminação de recursões à esquerda em G, obtém-se G', cujas produções formam o conjunto seguinte:

$$\begin{aligned} \{X_1 \rightarrow\ & X_2X_3 \mid a, \\ X_2 \rightarrow\ & X_3X_2 \mid b \mid X_3X_2X_2' \mid bX_2', \\ X_2' \rightarrow\ & bX_2' \mid b, \\ X_3 \rightarrow\ & X_4X_1 \mid c, \\ X_4 \rightarrow\ & cX_2X_3X_3 \mid bX_3X_3 \mid cX_2X_2'X_3X_3 \mid bX_2'X_3X_3 \mid aX_3 \mid ce \mid d \mid cX_2X_3X_3X_4' \mid bX_3X_3X_4' \mid \\ & cX_2X_2'X_3X_3X_4' \mid bX_2'X_3X_3X_4' \mid aX_3X_4' \mid ceX_4' \mid dX_4', \\ X_4' \rightarrow\ & X_1X_2X_3X_3 \mid X_1X_2X_2'X_3X_3 \mid X_1e \mid X_1X_2X_3X_3X_4' \mid X_1X_2X_2'X_3X_3X_4' \mid X_1eX_4'\} \end{aligned}$$

Note-se que, em G', todas as produções referentes a cada não-terminal $X_i \in N'$ se iniciam, conforme o esperado, com $\sigma \in \Sigma$, ou $X_j \in N$, $j > i$. Note-se ainda que todas as produções de X_4 são iniciadas exclusivamente com $\sigma \in \Sigma$. □

Tendo visto que toda gramática livre de contexto pode ser transformada em uma gramática equivalente, isenta de recursões à esquerda, pode-se passar finalmente ao algoritmo/teorema que possibilita a normalização em relação à Forma Normal de Greibach.

Teorema 4.10 (Forma Normal de Greibach) *Toda linguagem livre de contexto L pode ser gerada por uma gramática livre de contexto na Forma Normal de Greibach.*

Justificativa Pode ser desenvolvida a partir do Algoritmo 4.8, que permite a conversão de uma gramática livre de contexto qualquer em outra equivalente na Forma Normal de Greibach.

Algoritmo 4.8 (Forma Normal de Greibach) *Obtenção de uma gramática livre de contexto na Forma Normal de Greibach.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$ isenta de produções unitárias, símbolos inacessíveis, símbolos inúteis, produções em vazio e recursões à esquerda, tal que $L = L(G)$.

- Saída: uma gramática livre de contexto $G' = (V', \Sigma, P', S)$ na Forma Normal de Greibach, tal que $L(G) = L(G')$.
- Método:

1. Considere-se uma ordenação dos símbolos $X_i \in N$, tal que, se $X_i \rightarrow X_j\alpha, X_j \in N$ e $\alpha \in V^*$, então $j > i$. Seja, portanto, $X_1 < X_2 < \ldots < X_r$.
2. $N' \leftarrow N$;
3. $P' \leftarrow \{X_r \rightarrow \beta \mid X_r \rightarrow \beta \in P\}$;
4. $i \leftarrow r - 1$;
5. Se $i = 0$ então desviar para (6). Caso contrário:
   a) Considerem-se todas as produções de X_i iniciadas com X_j, $j > i$. Substitua-se X_j pela sua definição e repita-se este passo até que todas as alternativas de substituição para X_i sejam iniciadas apenas com símbolos terminais;
   b) $i \leftarrow i - 1$;
   c) Desviar para (5);
6. Para cada produção da forma:

$$X_i \rightarrow \sigma A_2 \ldots A_k, \text{ com } \sigma \in \Sigma, A_j \in V, 2 \leq j \leq k$$

   Se $A_j \in \Sigma$, substitua-se A_j por um novo não-terminal A'_j e faça-se $N' \leftarrow N \cup \{A'_j\}$, obtendo-se assim $X_i \rightarrow \sigma B_2 B_3 \ldots B_k$, com $B_j \in \{A_j, A'_j\}, 2 \leq j \leq k$. Incluir em P' todas as produções assim obtidas.
7. Para cada novo símbolo não-terminal A'_j introduzido no passo anterior, criar uma nova produção $A'_j \rightarrow A_j$ e fazer $P' \leftarrow P \cup \{A'_j \rightarrow A_j\}$.

Com relação ao passo (1), note-se que a condição de que G seja não-recursiva à esquerda garante que seja sempre possível estabelecer uma relação de ordem entre os elementos de N. Em particular, pode-se utilizar a própria relação de ordem obtida da aplicação do algoritmo de eliminação de recursões à esquerda. Observe-se, no passo (3), que β inicia necessariamente com um símbolo terminal.

Quanto aos novos símbolos não-terminais X'_i criados pela aplicação do referido método, pode-se, com segurança, atribuir aos mesmos os índices mais baixos nesta relação de ordem, situando-os antes dos demais símbolos X_i e com qualquer relação mútua de ordem (por exemplo, $X'_1 < \ldots < X'_q < X_1 < \ldots < X_p$) . Isso pode ser feito uma vez que, por construção, todas as produções de X'_i são iniciadas exclusivamente com $\sigma \in \Sigma$ ou $X_i \in N$.

No passo (5), cada produção de cada não-terminal X_i é processada através de substituições sucessivas até que ela se inicie exclusivamente com símbolos terminais.

Como há r não-terminais em N, a obtenção desta condição para $X_i \in N$, $1 \leq i \leq r$, exigirá no máximo $r - i$ substituições sucessivas. Ao término deste passo, todas as alternativas de substituição para X_i serão do tipo $\sigma\alpha$, $\sigma \in \Sigma$ e $\alpha \in V^*$.

Finalmente, para fazer com que α pertença ao conjunto N^*, conforme exigido pela Forma Normal de Greibach, basta criar novos símbolos não-terminais e substituí-los pelos respectivos símbolos terminais, onde quer que estes apareçam em $\alpha \in V^*$ — passo (6). Naturalmente, novas produções também devem ser incorporadas a P' — passo (7). ■

Exemplo 4.26 Retornando ao Exemplo 4.23, anteriormente apresentado:

$$\begin{aligned}
\{E &\rightarrow T \mid TE', \\
E' &\rightarrow +T \mid +TE', \\
T &\rightarrow F \mid FT', \\
T' &\rightarrow *F \mid *FT', \\
F &\rightarrow (E) \mid a\}
\end{aligned}$$

Considerando-se a relação de ordem $E' < T' < E < T < F$, obtém-se:

$$\begin{aligned}
F &\rightarrow (E) \mid a \\
T &\rightarrow (E) \mid a \mid (E)T' \mid aT' \\
E &\rightarrow (E) \mid a \mid (E)T' \mid aT' \mid (E)E' \mid aE' \mid (E)T'E' \mid aT'E' \\
T' &\rightarrow *F \mid *FT' \\
E' &\rightarrow +T \mid +TE'
\end{aligned}$$

Para finalizar, basta criar o não-terminal $)'$ e introduzir a produção $)' \rightarrow)$. A gramática resultante será:

$$\begin{aligned}
\{F &\rightarrow (E)' \mid a, \\
T &\rightarrow (E)' \mid a \mid (E)'T' \mid aT', \\
E &\rightarrow (E)' \mid a \mid (E)'T' \mid aT' \mid (E)'E' \mid aE' \mid (E)'T'E' \mid aT'E', \\
T' &\rightarrow *F \mid *FT', \\
E' &\rightarrow +T \mid +TE', \\
)' &\rightarrow)\}
\end{aligned}$$

□

Exemplo 4.27 No que se refere ao Exemplo 4.25, a obtenção da correspondente Forma Normal de Greibach, com base na relação de ordem $X_2' < X_4' < X_1 < X_2 < X_3 < X_4$, passa, inicialmente, pela substituição da definição de X_4 em X_3, de X_3 em X_2, de X_2 em X_1 e, finalmente, de X_1 em X_4'. À exceção deste último passo, omitido para não estender demais o exemplo, o resultado é apresentado a seguir. Observe-se ainda a criação do novo símbolo não-terminal "e'", com todas as ocorrências do símbolo "e" substituídas por "e'", e a adição ao conjunto de produções da regra $e' \rightarrow e$.

$$\begin{aligned}
\{X_1 \rightarrow\ & cX_2X_3X_3X_1X_2X_3 \mid bX_3X_3X_1X_2X_3 \mid cX_2X_2'X_3X_3X_1X_2X_3 \mid \\
& bX_2'X_3X_3X_1X_2X_3 \mid aX_3X_1X_2X3 \mid ce'X_1X_2X_3 \mid dX_1X_2X_3 \mid \\
& cX_2'X_3X_3X_4'X_1X_2X_3 \mid bX_3X_3X_4'X_1X_2X_3 \mid cX_2X_2'X_3X_3X_4'X_1X_2X_3 \mid \\
& bX_2'X_3X_3X_4'X_1X_2X_3 \mid aX_3X_4'X_1X_2X_3 \mid ce'X_4'X_1X_2X_3 \mid \\
& dX_4'X_1X_2X_3 \mid cX_2X_3 \mid cX_2X_3X_3X_1X_2X_2'X_3 \mid bX_3X_3X_1X_2X_2'X_3 \\
& cX_2X_2'X_3X_3X_1X_2X_2'X_3 \mid bX_2'X_3X_3X_1X_2X_2'X_3 \mid aX_3X_1X_2X_2'X_3 \mid
\end{aligned}$$

$$
\begin{array}{rcl}
 & & ce'X_1X_2X_2'X_3 \mid dX_1X_2X_2'X_3 \mid cX_2X_3X_3X_4'X_1X_2X_2'X_3 \mid \\
 & & bX_3X_3X_4'X_1X_2X_2'X_3 \mid cX_2X_2'X_3X_3X_4'X_1X_2X_2'X_3 \mid \\
 & & bX_2'X_3X_3X_4'X_1X_2X_2'X_3 \mid aX_3X_4'X_1X_2X_2'X_3 \mid ce'X_4'X_1X_2X_2'X_3 \mid \\
 & & dX_4'X_1X_2X_2'X_3 \mid cX_2X_2'X_3 \mid bX_2'X_3 \mid bX_3 \mid a, \\
X_2 & \rightarrow & cX_2X_3X_3X_1X_2 \mid bX_3X_3X_1X_2 \mid cX_2X_2'X_3X_3X_1X_2 \mid \\
 & & bX_2'X_3X_3X_1X_2 \mid aX_3X_1X_2 \mid ce'X_1X_2 \mid dX_1X_2 \mid \\
 & & cX_2X_3X_3X_4'X_1X_2 \mid bX_3X_3X_4'X_1X_2 \mid cX_2X_2'X_3X_3X_4'X_1X_2 \mid \\
 & & bX_2'X_3X_3X_4'X_1X_2 \mid aX_3X_4'X_1X_2 \mid ce'X_4'X_1X_2 \mid dX_4'X_1X_2 \mid cX_2 \mid \\
 & & cX_2X_3X_3X_1X_2X_2' \mid bX_3X_3X_1X_2X_2' \mid cX_2X_2'X_3X_3X_1X_2X_2' \mid \\
 & & bX_2'X_3X_3X_1X_2X_2' \mid aX_3X_1X_2X_2' \mid ce'X_1X_2X_2' \mid dX_1X_2X_2' \mid \\
 & & cX_2X_3X_3X_4'X_1X_2X_2' \mid bX_3X_3X_4'X_1X_2X_2' \mid cX_2X_2'X_3X_3X_4'X_1X_2X_2' \mid \\
 & & bX_2'X_3X_3X_4'X_1X_2X_2' \mid aX_3X_4'X_1X_2X_2' \mid ce'X_4'X_1X_2X_2' \mid \\
 & & dX_4'X_1X_2X_2' \mid cX_2X_2' \mid bX_2' \mid b, \\
X_2' & \rightarrow & bX_2' \mid b, \\
X_3 & \rightarrow & cX_2X_3X_3X_1 \mid bX_3X_3X_1 \mid cX_2X_2'X_3X_3X_1 \mid bX_2'X_3X_3X_1 \mid aX_3X_1 \mid \\
 & & ce'X_1 \mid dX_1 \mid cX_2X_3X_3X_4'X_1 \mid bX_3X_3X_4'X_1 \mid cX_2X_2'X_3X_3X_4'X_1 \mid \\
 & & bX_2'X_3X_3X_4'X_1 \mid aX_3X_4'X_1 \mid ce'X_4'X_1 \mid dX_4'X_1 \mid c, \\
X_4 & \rightarrow & cX_2X_3X_3 \mid bX_3X_3 \mid cX_2X_2'X_3X_3 \mid bX_2'X_3X_3 \mid aX_3 \mid ce' \mid d \mid \\
 & & cX_2X_3X_3X_4' \mid bX_3X_3X_4' \mid cX_2X_2'X_3X_3X_4' \mid \\
 & & bX_2'X_3X_3X_4' \mid aX_3X_4' \mid ce'X_4' \mid dX_4', \\
X_4' & \rightarrow & X_1X_2X_3X_3 \mid X_1X_2X_2'X_3X_3 \mid X_1e' \mid X_1X_2X_3X_3X_4' \mid X_1X_2X_2'X_3X_3X_4' \mid X_1e'X_4', \\
e' & \rightarrow & e\}
\end{array}
$$

□

4.7 Autômatos de Pilha

Nesta seção serão apresentados os conceitos básicos ligados à representação e à interpretação dos autômatos de pilha, sendo o principal a definição de dois modelos de aceitação distintos, um baseado em estados finais e outro em pilha vazia.

A seguir, é mostrada a equivalência da classe de linguagens aceita pelos autômatos de pilha com critério de aceitação baseado em estados finais, com a classe de linguagens aceita pelos autômatos de pilha cujo critério é baseado em pilha vazia, evidenciando assim a equipotência dos modelos.

A equivalência da classe de linguagens aceita pelos autômatos de pilha com a classe de linguagens gerada por gramáticas do tipo 2 — as linguagens livres de contexto — será discutida na Seção 4.8. Por último, serão efetuadas algumas considerações sobre a aplicação e a importância prática da teoria dos autômatos de pilha para o profissional de computação.

Os **autômatos de pilha** constituem a segunda instância, em uma escala de complexidade crescente, do modelo genérico de reconhecedor introduzido na Seção 2.5, prestando-se ao reconhecimento de linguagens livres de contexto. Diferentemente dos autômatos finitos, que não se utilizam da memória auxiliar prevista no modelo genérico, os autômatos de pilha têm o seu poder de reconhecimento estendido, quando comparado ao dos autômatos finitos, justamente pela disponibilidade e pela utilização de uma memória auxiliar organizada na forma de uma pilha.

A pilha de um autômato de pilha pode ser entendida como uma estrutura de dados, de capacidade ilimitada, na qual a máquina de estados é capaz de armazenar, consultar e remover símbolos de um alfabeto próprio, denominado **alfabeto de pilha**, segundo a convenção usual para estruturas deste tipo (LIFO — "last-in-first-out"). Quanto aos seus demais componentes e características, o autômato de pilha se assemelha ao autômato finito anteriormente estudado.

Formalmente, um autômato de pilha pode ser definido como uma sétupla M:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

onde:

Q	é um conjunto finito de estados;
Σ	é um alfabeto (finito e não-vazio) de entrada;
Γ	é um alfabeto (finito e não-vazio) de pilha;
δ	é uma função de transição $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$;
q_0	é o estado inicial de M, $q_0 \in Q$;
Z_0	é o símbolo inicial da pilha, $Z_0 \in \Gamma$;
F	é o conjunto de estados finais de M, $F \subseteq Q$.

Dos sete elementos que compõem este sistema formal, apenas três merecem explicação adicional — δ, Γ e Z_0 —, uma vez que os demais estão em correspondência direta com o que já foi estudado para o caso dos autômatos finitos.

Note-se, inicialmente, a presença de um alfabeto de pilha Γ e de um símbolo inicial de pilha Z_0. O alfabeto de pilha especifica os símbolos que podem ser armazenados na pilha. Por convenção, um desses símbolos, denotado Z_0, representa o conteúdo inicial da pilha toda vez que o autômato de pilha principia o reconhecimento de uma nova sentença. Ao longo de sua operação, elementos de Γ são acrescentados e/ou removidos da pilha, e seu conteúdo total pode ser interpretado, em um dado instante, como sendo um elemento de Γ^*. Por convenção, cadeias de Γ que representam o conteúdo da pilha em um determinado instante são interpretadas considerando-se os símbolos mais à esquerda da cadeia no topo da pilha, e os símbolos mais à direita da cadeia no fundo da pilha.

Observe-se que, a partir do formato geral apresentado acima para a função de transição δ em autômatos de pilha não-determinísticos e com transições em vazio, é possível extrair os seguintes casos particulares, equivalentes aos estudados do Capítulo 3, para os autômatos finitos:

- Autômato de pilha determinístico sem transições em vazio:
  $\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$

- Autômato de pilha determinístico com transições em vazio:
  $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$

- Autômato de pilha não-determinístico sem transições em vazio:
  $Q \times \Sigma \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$

- Autômato de pilha não-determinístico com transições em vazio:
  $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$

Diferentemente do Capítulo 3, em que os modelos de autômato finito foram considerados em escala crescente de complexidade, os autômatos de pilha serão considerados, no presente capítulo, a partir de seu modelo mais geral, ficando para o leitor a tarefa de interpretar e eventualmente utilizar, onde adequado, os modelos mais restritos pre-

viamente relacionados, sem que isso implique a perda de validade dos conceitos e dos resultados a seguir apresentados.

Exemplo 4.28 Considere-se $\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$. A cadeia $\gamma_2\gamma_1\gamma_4\gamma_4$, por exemplo, representa o conteúdo da pilha conforme ilustrado na Figura 4.5. Note-se que γ_2 se encontra no topo da pilha e que γ_4 se encontra no fundo da mesma:

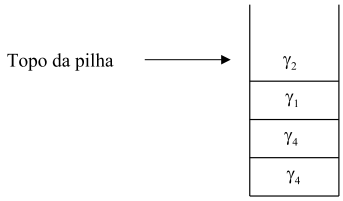

Figura 4.5: Pilha com ponteiro de topo

□

Os dispositivos que fazem uso de pilha costumam definir a forma de operação da mesma como uma das seguintes possibilidades:

- **Pilha "Stack"**: além das operações de empilhamento e desempilhamento de elementos no topo da pilha ("push" e "pop"), permite que os demais elementos da mesma sejam endereçados diretamente, somente para consulta;

- **Pilha "Pushdown"**: permite o acesso apenas ao elemento armazenado no topo da pilha, através das operações de empilhamento e desempilhamento ("push" e "pop"). Não permite o endereçamento dos demais elementos da pilha.

A definição de autômato de pilha neste texto considera que a sua memória auxiliar seja uma pilha "pushdown".

A **configuração de um autômato de pilha** é definida pelo seu estado corrente, pela parte da cadeia de entrada ainda não analisada e pelo conteúdo da pilha. A **configuração inicial de um autômato de pilha** é aquela em que o autômato se encontra no estado inicial q_0, o cursor se encontra posicionado sob a célula mais à esquerda na fita de entrada e o conteúdo da pilha é Z_0. Algebricamente, a configuração de um autômato de pilha pode ser representada como uma tripla:

$$(q, \alpha, \gamma), \text{ com } q \in Q, \alpha \in \Sigma^* \text{ e } \gamma \in \Gamma^*$$

A configuração inicial para o reconhecimento de uma cadeia w é representada como (q_0, w, Z_0).

As possibilidades de movimentação de um autômato de pilha em uma dada configuração são determinadas a partir de três informações: o seu estado corrente, o próximo símbolo presente na cadeia de entrada e o símbolo armazenado no topo da pilha. Observe-se, pela definição da função δ, a possibilidade de movimentações em vazio, sem consumo de símbolos da fita de entrada, e também a possibilidade de serem especificadas transições não-determinísticas. Note-se também a obrigatoriedade, imposta por essa formulação, de se consultar o símbolo presente no topo da pilha em toda e qualquer transição efetuada pelo autômato.

Após a aplicação de uma transição, o cursor de leitura sofre um deslocamento de uma posição para a direita, e o símbolo presente no topo da pilha é removido, sendo substituído pela cadeia de símbolos especificada no lado direito da transição. No caso de

transições em vazio, em que não há consulta de símbolo na fita de entrada, a posição do cursor permanece inalterada após a sua aplicação.

Desejando-se efetuar alguma transição de forma independente do conteúdo do topo da pilha, torna-se necessário especificar para cada elemento do alfabeto de pilha uma transição que efetue o mesmo tratamento do símbolo de entrada, removendo e reinserindo o mesmo símbolo da pilha, simulando assim uma transição independente do conteúdo da pilha.

Para cada tripla (q, σ, γ) pertencente ao domínio da função δ, o elemento $\delta(q, \sigma, \gamma)$ pode conter zero, um ou mais elementos de $Q \times \Gamma^*$. Considere-se inicialmente o caso em que $\sigma \neq \epsilon$. Havendo zero elementos em $\delta(q, \sigma, \gamma)$, isso indica que não há possibilidade de movimentação a partir da configuração considerada. Havendo um único elemento, isso significa que há apenas uma possibilidade de movimentação e, portanto, a transição é determinística. Quando todas as transições de um autômato de pilha são determinísticas, diz-se que o mesmo é **determinístico**.

Nos casos em que há mais de um elemento em $\delta(q, \sigma, \gamma)$, isso significa que há mais de uma opção de movimentação a partir desta configuração, e então a transição é dita não-determinística. Havendo pelo menos uma transição não-determinística, diz-se que o autômato de pilha é **não-determinístico**.

Autômatos de pilha não-determinísticos e com transições em vazio apresentam um comportamento dinâmico determinístico quando as duas seguintes condições forem simultaneamente verificadas:

1. $\forall\, q \in Q$, $\gamma \in \Gamma$, se $\left|(\delta, q, \epsilon, \gamma)\right| \geq 1$, então $\left|\delta(q, \sigma, \gamma)\right| = 0$, $\forall\, \sigma \in \Sigma$;
2. $\forall\, q \in Q$, $\gamma \in \Gamma$, $\sigma \in \Sigma \cup \{\epsilon\}$, $\left|\delta(q, \sigma, \gamma))\right| \leq 1$.

A primeira condição estabelece que não deve existir, para uma mesma combinação de estado corrente e símbolo no topo da pilha, a opção de escolha entre uma movimentação com consumo de símbolo na cadeia de entrada e outra sem consumo de símbolo (transição em vazio). A segunda condição estabelece que, para cada configuração, seja ela qual for, não deve haver mais de uma opção distinta de movimentação.

Surge, neste ponto, um aspecto de grande relevância para o estudo dos autômatos de pilha e das linguagens livres de contexto. Diferentemente do que foi visto para o caso dos autômatos finitos, no que se refere à equivalência dos modelos determinístico e não-determinístico quanto à classe de linguagens que são capazes de reconhecer, os autômatos de pilha não apresentam correspondente equivalência. Conforme será visto mais adiante, os autômatos de pilha determinísticos são capazes de reconhecer apenas um subconjunto das linguagens livres de contexto. Por esse motivo, exceto por ressalvas em sentido contrário, os autômatos de pilha mencionados daqui em diante serão não-determinísticos.

Estando o autômato de pilha em uma configuração $(q_i, \sigma\alpha, \phi\gamma)$, com $q_i \in Q$, $\sigma \in \Sigma$, $\alpha \in \Sigma^*$, $\phi \in \Gamma$ e $\gamma \in \Gamma^*$, sua movimentação a partir dessa configuração para a seguinte, como decorrência da aplicação de uma transição $\delta(q_i, \sigma, \phi) = (q_j, \eta)$, com $\eta \in \Gamma^*$, é representada por:

$$(q_i, \sigma\alpha, \phi\gamma) \vdash (q_j, \alpha, \eta\gamma)$$

A aplicação de transições em vazio, em que não há consumo de símbolo, é representada de maneira similar através de:

$$(q_i, \alpha, \phi\gamma) \vdash (q_j, \alpha, \eta\gamma)$$

Da mesma forma que no caso dos autômatos finitos, a aplicação de zero ou mais transições entre duas configurações quaisquer é representada através do símbolo $\vdash^*$, e a aplicação de no mínimo uma transição, por intermédio do símbolo $\vdash^+$.

A **configuração final de um autômato de pilha** costuma ser caracterizada de duas maneiras distintas, porém equivalentes. Na primeira delas, exige-se o esgotamento da cadeia de entrada e também que o autômato atinja um estado final. Nesta caracterização, o conteúdo final da pilha é irrelevante. Na segunda caracterização, exige-se o esgotamento da cadeia de entrada e também que a pilha tenha sido completamente esvaziada, não importando que o estado atingido seja final ou não-final.

Dependendo do tipo de definição adotada para caracterizar a configuração final de um autômato de pilha, é possível definir, também de duas maneiras distintas, a linguagem aceita pelo dispositivo.

A linguagem aceita por um autômato de pilha M, com base no **critério de estado final**, denotada $L(M)$, é definida como sendo o conjunto das sentenças w que satisfazem à condição:

$$(q_0, w, Z_0) \vdash^* (q, \epsilon, \gamma), q \in F, \gamma \in \Gamma^*$$

Analogamente, a linguagem aceita por um autômato de pilha M, com base no **critério de pilha vazia**, denotada $V(M)$, é definida como sendo o conjunto das sentenças w que satisfazem à condição:

$$(q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon), q \in Q$$

Assim como no caso dos autômatos finitos, convém lembrar que uma dada cadeia w é aceita por um autômato de pilha se e apenas se o autômato for capaz de atingir uma configuração final, conforme o critério estabelecido para o reconhecedor, a partir de sua configuração inicial. No caso de autômatos de pilha determinísticos, a seqüência de movimentos que os conduzem de uma configuração a outra, se existir, deve ser única. Sendo possível atingir alguma configuração final, diz-se que a cadeia pertence à linguagem definida pelo autômato. Caso contrário, que ela não pertence à linguagem aceita pelo autômato de pilha.

Autômatos de pilha não-determinísticos, ao contrário, necessitam, no pior caso, percorrer exaustivamente todas as seqüências possíveis de configurações, tendo a cadeia de entrada w como referência, até que seja possível concluir pela sua aceitação ou rejeição. O simples fato de não se conseguir atingir uma configuração final ao término de uma série de movimentações só poderá ser considerado como indicativo da rejeição da cadeia de entrada se não houver novas seqüências de configurações a serem experimentadas.

Com o intuito de facilitar a leitura do texto no restante deste capítulo, as transições de um autômato de pilha serão denotadas como:

$$(q_i, \sigma, X) \rightarrow (q_j, \gamma)$$

indicando, com isso, o par ordenado $((q_i, \sigma, X), (q_j, \gamma))$ pertencente à função δ, ou, simplesmente, $\delta(q_i, \sigma, X) = (q_j, \gamma)$.

Exemplo 4.29 Seja M_1 o autômato de pilha determinístico abaixo discriminado, cujo critério de aceitação de sentenças é baseado no esvaziamento da pilha.

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{a, b, c\} \\ \Gamma &= \{Z_0, C\} \end{aligned}$$

$$\begin{aligned}\delta &= \{(q_0, a, Z_0) \rightarrow \{(q_0, CCZ_0)\},\\ &\quad (q_0, a, C) \rightarrow \{(q_0, CCC)\},\\ &\quad (q_0, b, Z_0) \rightarrow \{(q_1, Z_0)\},\\ &\quad (q_0, b, C) \rightarrow \{(q_1, C)\},\\ &\quad (q_1, c, C) \rightarrow \{(q_1, \epsilon)\},\\ &\quad (q_1, \epsilon, Z_0) \rightarrow \{(q_1, \epsilon)\}\}\\ F &= \varnothing\end{aligned}$$

Deve-se notar, em primeiro lugar, que o determinismo deste autômato decorre do fato de que a função δ exibe no máximo um elemento para cada combinação possível de estado, símbolo de entrada e símbolo no topo da pilha, e também porque inexistem transições em vazio que se refiram a pares de estado e símbolo no topo da pilha considerados nas transições não-vazias: note-se que a única transição em vazio deste autômato ocorre no estado q_1, com um símbolo do alfabeto de pilha (Z_0) diferente do utilizado na transição não-vazia desse mesmo estado (C).

A linguagem definida por M_1 é:

$$V_1(M_1) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon), q \in Q\}$$

Inspecione-se agora as produções deste autômato com o intuito de determinar a linguagem por ele aceita. Partindo-se do estado inicial, os únicos símbolos de entrada admitidos são a e b. Para cada símbolo a consumido na entrada, a pilha é acrescida da cadeia CC em seu topo, preservando-se o restante de seu conteúdo. Ao ocorrer uma transição com o símbolo de entrada b, há apenas uma mudança de estado, não havendo qualquer alteração no conteúdo da pilha. Nesse novo estado, note-se que são especificadas apenas transições envolvendo o símbolo de entrada c. Para cada c consumido, um símbolo C é retirado do topo da pilha, até que reste apenas o símbolo inicial da pilha, o qual é removido através de uma transição em vazio.

Observe-se que o conjunto de produções assim formulado impõe que as sentenças aceitas pelo autômato sejam construídas de modo tal que, para cada símbolo a previamente consumido no início da cadeia, dois símbolos c sejam necessariamente colocados em correspondência com ele no final da cadeia de entrada. Note-se ainda que a seqüência de símbolos a é separada da seqüência de símbolos c por exatamente um símbolo b. Observe-se que o papel desempenhado pela pilha, neste caso, consiste em manter a "memória" da quantidade de símbolos a que foram localizados no início da cadeia, possibilitando o reconhecimento futuro da correspondente quantidade de símbolos c ao término da cadeia de entrada.

Como conseqüência, o conjunto de sentenças capazes de conduzir este autômato a configurações finais, com base no critério de pilha vazia, e portanto a linguagem por ele definida, é o seguinte:

$$V_1(M_1) = \{a^i bc^{2i} \mid i \geq 0\}$$

Considerem-se algumas sentenças desta linguagem e a correspondente seqüência de movimentos executada pelo autômato durante o seu reconhecimento:

1. Sentença: b

   Movimentos: $(q_0, b, Z_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_1, \epsilon, \epsilon)$

2. Sentença: $abcc$

   Movimentos: $(q_0, abcc, Z_0) \vdash (q_0, bcc, CCZ_0) \vdash (q_1, cc, CCZ_0) \vdash (q_1, c, CZ_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_1, \epsilon, \epsilon)$

3. Sentença: $aabcccc$

   Movimentos: $(q_0, aabcccc, Z_0) \vdash (q_0, abcccc, CCZ_0) \vdash (q_0, bcccc, CCCCZ_0) \vdash (q_1, cccc, CCCCZ_0) \vdash (q_1, ccc, CCCZ_0) \vdash (q_1, cc, CCZ_0) \vdash (q_1, c, CZ_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_1, \epsilon, \epsilon)$

Tome-se agora como exemplo duas sentenças que não pertencem à linguagem definida por este autômato, e as correspondentes seqüências de configurações:

1. Sentença: $abccc$

   Movimentos: $(q_0, abccc, Z_0) \vdash (q_0, bccc, CCZ_0) \vdash (q_1, ccc, CCZ_0) \vdash (q_1, cc, CZ_0) \vdash (q_1, c, Z_0)$

2. Sentença: $aabccc$

   Movimentos: $(q_0, aabccc, Z_0) \vdash (q_0, abccc, CCZ_0) \vdash (q_0,\ bccc,\ CCCCZ_0) \vdash$
   $(q_1,\ ccc,\ CCCCZ_0) \vdash (q_1, cc, CCCZ_0) \vdash (q_1, c, CCZ_0) \vdash (q_1, \epsilon, CZ_0)$

No primeiro caso, a sentença $abccc$ contém símbolos c em excesso, e por esse motivo ela é rejeitada ao ser atingida a configuração (q_1, c, Z_0). No segundo caso, a sentença $aabccc$ apresenta quantidade insuficiente de símbolos c, e devido a isso ela é rejeitada na configuração (q_1, ϵ, CZ_0). Observe-se que ambas as configurações não admitem evolução, conforme a especificação da função δ, e além disso elas também não são configurações finais do autômato. □

Convém, neste ponto, introduzir uma extensão na notaçã dos Diagramas de Estado estudados para o caso dos autômatos finitos, com o intuito de permitir a representação gráfica também dos autômatos de pilha. Diferentemente dos Diagramas de Estado definidos para os autômatos finitos, os arcos entre dois estados p e q são rotulados com cadeias da forma:

$$(\sigma, Z)/\gamma,\ \text{com } \sigma \in \Sigma, Z \in \Gamma \text{ e } \gamma \in \Gamma^*$$

para cada produção $\delta(p, \sigma, Z) = (q, \gamma)$.

Exemplo 4.30 Considere-se o autômato de pilha não-determinístico M_2 a seguir apresentado, cujo critério de aceitação de sentenças é baseado em estado final.

$$\begin{aligned}
Q &= \{q_0, q_1, q_2, q_3, q_4\} \\
\Sigma &= \{a, c\} \\
\Gamma &= \{Z_0, C\} \\
\delta &= \{(q_0, a, Z_0) \to \{(q_1, CCZ_0), (q_2, CZ_0)\}, \\
&\quad (q_1, a, C) \to \{(q_1, CCC)\}, \\
&\quad (q_1, c, C) \to \{(q_3, \epsilon)\}, \\
&\quad (q_2, a, C) \to \{(q_2, CC)\}, \\
&\quad (q_2, c, C) \to \{(q_3, \epsilon)\}, \\
&\quad (q_3, c, C) \to \{(q_3, \epsilon)\}, \\
&\quad (q_3, \epsilon, Z_0) \to \{(q_4, Z_0)\}\} \\
F &= \{q_4\}
\end{aligned}$$

O não-determinismo deste autômato decorre do fato de que, na configuração inicial, existem duas possibilidades de movimentação quando o símbolo presente na cadeia de entrada for a. O fato de o critério de aceitação ser baseado em estado final não tem qualquer vínculo com o não-determinismo deste autômato, tendo sido introduzido apenas com o intuito de ilustrar a aplicação do critério. Conforme mostrado nos Teoremas 4.11 e 4.12, o critério de aceitação pode ser facilmente alterado sem prejuízo da linguagem sendo definida. Não obstante, o não-determinismo é essencial para a aceitação da linguagem definida por este autômato.

A linguagem definida por M_2 é:

$$L_2(M_2) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q_4, \epsilon, \epsilon)\}$$

A Figura 4.6 ilustra o Diagrama de Estados para o autômato M_2:

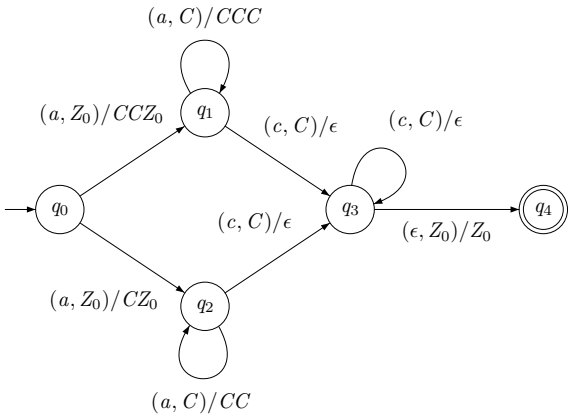

Figura 4.6: Autômato de pilha do Exemplo 4.30

A determinação da linguagem aceita por este autômato pode ser feita analisando-se separadamente a linguagem aceita a partir de cada um dos dois ramos em que o reconhecimento se subdivide após o não-determinismo inicial.

Para tanto, basta notar que o ramo superior deste autômato (estados q_0 e q_1) acrescenta dois símbolos "C" à pilha para cada símbolo "a" localizado na cadeia de entrada. O ramo inferior (estados q_0 e q_2), por sua vez, acrescenta apenas um símbolo "C" à pilha para cada símbolo "a" consumido. Daí para a frente, apenas símbolos "c" da cadeia de entrada serão considerados. Para cada "c" consumido, note-se que exatamente um símbolo "C" é removido da pilha. Finalmente, no estado q_3, caso todos os símbolos "C" da pilha já tenham sido eliminados, é possível transitar em vazio para o estado final q_4.

Como conclusão, tem-se que a configuração final será atingida apenas para os casos de cadeias em que a quantidade de símbolos "c" seja igual à quantidade de símbolos "a" que a antecede, ou então exatamente igual ao dobro da quantidade de símbolos "a". Formalmente,

$$L(M_2) = \{a^i c^j \mid i \geq 1 \text{ e } (j = i \text{ ou } j = 2i)\}$$

Observe-se que não há, *a priori*, como o autômato "descobrir" se ele deve estabelecer a correspondência de cada "a" encontrado com apenas um ou então dois "c" subseqüentes. Por esse motivo, o não-determinismo inicial oferece duas alternativas, cada qual associada a uma hipótese sobre a estrutura da cadeia a ser analisada. Havendo fracasso no reconhecimento iniciado por uma das alternativas, deve-se tentar a outra. Persistindo o fracasso, pode-se então dizer que a cadeia não pertence à linguagem definida.

A operação deste autômato é exemplificada a seguir através do reconhecimento das seguintes sentenças:

1. Sentença: $aacc$

   Movimentos: $(q_0, aacc, Z_0) \vdash (q_2, acc, CZ_0) \vdash (q_2, cc, CCZ_0) \vdash (q_3, c, CZ_0) \vdash (q_3, \epsilon, Z_0) \vdash (q_4, \epsilon, Z_0)$

2. Sentença: $aacccc$

   Movimentos: $(q_0, aacccc, Z_0) \vdash (q_1, acccc, CCZ_0) \vdash (q_1, cccc, CCCCZ_0) \vdash (q_3, ccc, CCCZ_0) \vdash (q_3, cc, CCZ_0) \vdash (q_3, c, CZ_0) \vdash (q_3, \epsilon, Z_0) \vdash (q_4, \epsilon, Z_0)$

Note-se que o reconhecimento, em ambos os casos, pôde ser bem-sucedido já na primeira seqüência de movimentos, uma vez que a escolha da transição a ser aplicada na configuração inicial foi corretamente "adivinhada" nas duas situações. No entanto, a escolha da transição a ser aplicada na configuração inicial do primeiro caso poderia ter sido diferente da apresentada, e neste caso ocorreria o seguinte:

1. Sentença: $aacc$

   Movimentos: $(q_0, aacc, Z_0) \vdash (q1, acc, CCZ_0) \vdash (q_1, cc, CCCCZ_0) \vdash (q_3, c, CCCZ_0) \vdash (q_3, \epsilon, CCZ_0)$

Como se pode perceber, atinge-se neste caso uma configuração não-final, em que não há possibilidade de evolução. Como conseqüência, por se tratar de um autômato não-determinístico, dever-se-ia retornar à configuração inicial e efetuar outra seqüência de movimentações, adotando-se a segunda alternativa, em lugar da primeira, para o caso da transição não-determinística (ou seja, exatamente o caso correto apresentado anteriormente).

Esquematicamente, as várias seqüências de configurações que podem ser percorridas em decorrência da existência de transições não-determinísticas podem ser representadas graficamente na forma de uma árvore, conforme ilustrado na Figura 4.7:

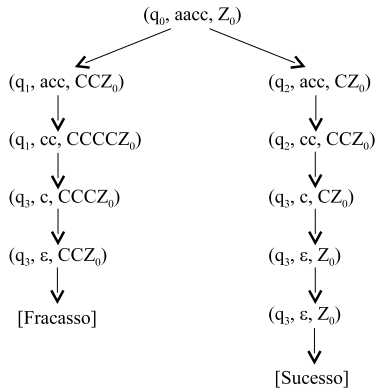

Figura 4.7: Aceitação em autômato de pilha não-determinístico do Exemplo 4.30

Considere-se, agora, o caso de uma cadeia — aac — que não pertença à linguagem definida por este autômato. Conforme ilustrado na Figura 4.8, a rejeição desta cadeia ocorre apenas após o fracasso do reconhecimento em todas as seqüências possíveis de movimentação (duas, para este autômato):

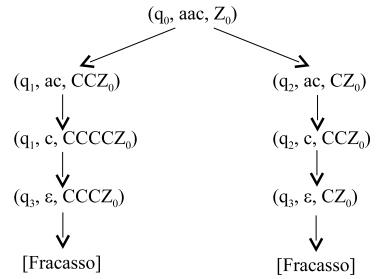

Figura 4.8: Rejeição em autômato de pilha não-determinístico do Exemplo 4.30

□

Para finalizar este item, cumpre notar que, de acordo com a definição, os autômatos de pilha são capazes de efetuar movimentos independentemente da existência de símbolos na fita de entrada, através das chamadas transições em vazio, ao passo que o esvaziamento da pilha necessariamente impede qualquer possibilidade de movimentação futura. Ambos estes fatos serão utilizados em seguida para demonstrar a equivalência dos critérios de aceitação por estado final e por pilha vazia.

A classe de linguagens aceita por autômatos de pilha não-determinísticos com critério de aceitação baseado em estado final é idêntica à classe de linguagens aceita por autômatos de pilha não-determinísticos com critério de aceitação baseado em pilha vazia.

A importância desse resultado deve-se à liberdade de escolha que ele oferece quando se pretende demonstrar algum teorema relativo aos autômatos de pilha e às linguagens livres de contexto, podendo-se optar indistintamente entre um e outro critério de aceitação, sem prejuízo para a sua generalização.

Em particular, este resultado será empregado na Seção 4.8 na demonstração da equivalência entre a classe de linguagens aceita pelos autômatos de pilha e a classe das linguagens livres de contexto. Naquele caso, a opção pelo critério de aceitação baseado no esvaziamento da pilha permite uma simplificação significativa da demonstração.

Teorema 4.11 (Estado final ⇒ pilha vazia) *Seja M um autômato de pilha não-determinístico com critério de aceitação baseado em estado final. Então, é possível definir um autômato de pilha não-determinístico M' com critério de aceitação baseado em pilha vazia, de modo que $L(M) = V(M')$.*

Justificativa O Algorimo 4.9 apresenta um método para se efetuar tal conversão.

Algoritmo 4.9 (Estado final ⇒ pilha vazia) *Obtenção de um autômato de pilha baseado em critério de aceitação de pilha vazia a partir de outro baseado em critério de estado final.*

- Entrada: um autômato de pilha não-determinístico $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, com critério de aceitação baseado em estado final.
- Saída: um autômato de pilha não-determinístico $M' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', \varnothing)$, com critério de aceitação baseado em pilha vazia.
- Método:

1. $Q' \leftarrow Q \cup \{q_v, q_0'\}$ (supondo-se, sem perda de generalidade, que $Q \cap \{q_v, q_0'\} = \varnothing$);
2. $\Gamma' \leftarrow \Gamma \cup \{Z_0'\}$ (supondo-se, sem perda de generalidade, que $\Gamma \cap \{Z_0'\} = \varnothing$);
3. Função de transição δ':

   a) $\delta' \leftarrow \varnothing$;

   b) $\delta'(q_0', \epsilon, Z_0') \leftarrow \{(q_0, Z_0 Z_0')\}$;

   c) $\delta'(q, \sigma, \phi) \leftarrow \delta(q, \sigma, \phi), \quad \forall\, q \in Q, \phi \in \Gamma, \sigma \in (\Sigma \cup \{\epsilon\})$;

   d) $\delta'(q, \epsilon, \phi) \leftarrow \delta'(q, \epsilon, \phi) \cup \{(q_v, \epsilon)\}, \quad \forall\, q \in F, \phi \in (\Gamma \cup \{Z_0'\})$;

   e) $\delta'(q_v, \epsilon, \phi) \leftarrow \{(q_v, \epsilon)\}, \quad \forall\, \phi \in (\Gamma \cup \{Z_0'\})$.

O princípio básico que justifica a formulação acima apresentada consiste em fazer M' simular M e fazer com que as sentenças que levam M a atingir um estado final levem

M' a esvaziar a sua pilha. Dessa maneira, se M atingir uma configuração final, M' também a atingirá.

No entanto, alguns cuidados especiais devem ser tomados para garantir que não haja sentenças simultaneamente aceitas por M' e rejeitadas por M. Observe que M' possui dois estados a mais que M, e também um elemento a mais no alfabeto de pilha. O símbolo Z_0, que é o símbolo inicial de pilha em M, é inserido na pilha através da transição descrita no passo (3.b) do algoritmo. O estado inicial q_0' foi criado especialmente para permitir a inserção de Z_0 no fundo da pilha logo ao início do reconhecimento. Depois disso, as transições descritas no passo (3.c) fazem com que M' simule o comportamento de M. Note-se que Z_0', o símbolo inicial de pilha em M', permanece intacto no fundo da pilha até que seja atingido algum estado final de M.

A intenção de se manter Z_0' no fundo da pilha ao longo do reconhecimento é dupla. Por um lado, visa garantir que, na eventualidade de M atingir um estado não-final com a sua pilha vazia, não haverá risco de a correspondente cadeia ser aceita por M' (obviamente, ela é rejeitada por M). Por outro lado, quando M atinge um estado final, visa garantir que sempre haverá pelo menos um símbolo na pilha, permitindo assim a transição de M' para o estado final q_v, o que é feito através das transições do passo (3.d) — é preciso lembrar que autômatos de pilha dependem da presença de símbolos no topo da pilha para poderem evoluir em sua configuração.

No estado q_v a pilha é esvaziada, por meio das transições do passo (3.d), sem consumir símbolos da fita de entrada. Assim, toda e qualquer cadeia que seja totalmente consumida por M, fazendo com que o mesmo atinja um estado final — ou seja, toda e qualquer cadeia aceita por M —, fará com que M' atinja o estado q_v, também com a cadeia de entrada esgotada, porém com a pilha vazia — ou seja, será aceita por M'. Logo, M e M' aceitam a mesma linguagem. ■

Exemplo 4.31 Considere-se o autômato de pilha M:

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{a, b, c\} \\
\Gamma &= \{Z_0, A, B\} \\
\delta &= \{(q_0, a, Z_0) \to \{(q_0, AZ_0)\}, (q_1, a, A) \to \{(q_1, \epsilon)\}, \\
&\quad (q_0, a, A) \to \{(q_0, AA)\}, (q_1, b, B) \to \{(q_1, \epsilon)\}, \\
&\quad (q_0, a, B) \to \{(q_0, AB)\}, (q_1, \epsilon, Z_0) \to \{(q_2, \epsilon)\}, \\
&\quad (q_0, b, Z_0) \to \{(q_0, BZ_0)\}, (q_0, b, A) \to \{(q_0, BA)\}, \\
&\quad (q_0, b, B) \to \{(q_0, BB)\}, (q_0, c, Z_0) \to \{(q_2, \epsilon)\}, \\
&\quad (q_0, c, A) \to \{(q_1, A)\}, (q_0, c, B) \to \{(q_1, B)\}\} \\
F &= \{q_2\}
\end{aligned}
$$

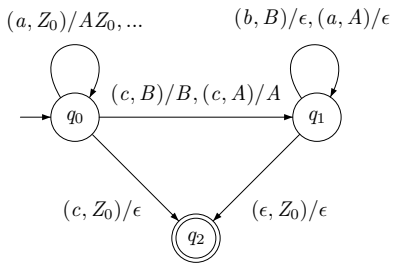

Figura 4.9: Autômato M com critério de aceitação de estado final

A Figura 4.9 ilustra M na notação dos Diagramas de Estados. A linguagem definida por ele é $\{\alpha c \alpha^R \mid \alpha \in \{a, b\}^*\}$. Aplicando-se o algoritmo de conversão do critério de aceitação, obtém-se o seguinte autômato M' equivalente, com aceitação baseada no esvaziamento da pilha:

$$
\begin{aligned}
M' &= (Q', \Sigma, \Gamma', \delta', q_0', Z_0', \varnothing) \\
Q' &= \{q_0, q_1, q_2, q_v, q_0'\} \\
\Gamma' &= \{Z_0, A, B, Z_0'\} \\
\delta' &= \{(q_0, a, Z_0) \rightarrow \{(q_0, AZ_0)\}, (q_1, a, A) \rightarrow \{(q_1, \epsilon)\}, \\
&\quad (q_0, a, A) \rightarrow \{(q_0, AA)\}, (q_1, b, B) \rightarrow \{(q_1, \epsilon)\}, \\
&\quad (q_0, a, B) \rightarrow \{(q_0, AB)\}, (q_1, \epsilon, Z_0) \rightarrow \{(q_2, \epsilon)\}, \\
&\quad (q_0, b, Z_0) \rightarrow \{(q_0, BZ_0)\}, (q_0, b, A) \rightarrow \{(q_0, BA)\}, \\
&\quad (q_0, b, B) \rightarrow \{(q_0, BB)\}, (q_0, c, Z_0) \rightarrow \{(q_2, \epsilon)\}, \\
&\quad (q_0, c, A) \rightarrow \{(q_1, A)\}, (q_0, c, B) \rightarrow \{(q_1, B)\}, \\
&\quad (q_0', \epsilon, Z_0') \rightarrow \{(q_0, Z_0 Z_0')\}, \\
&\quad (q_2, \epsilon, Z_0') \rightarrow \{(q_v, \epsilon)\}, \\
&\quad (q_2, \epsilon, Z_0) \rightarrow \{(q_v, \epsilon)\}, \\
&\quad (q_2, \epsilon, A) \rightarrow \{(q_v, \epsilon)\}, \\
&\quad (q_2, \epsilon, B) \rightarrow \{(q_v, \epsilon)\}, \\
&\quad (q_v, \epsilon, Z_0') \rightarrow \{(q_v, \epsilon)\}, \\
&\quad (q_v, \epsilon, Z_0) \rightarrow \{(q_v, \epsilon)\}, \\
&\quad (q_v, \epsilon, A) \rightarrow \{(q_v, \epsilon)\}, \\
&\quad (q_v, \epsilon, B) \rightarrow \{(q_v, \epsilon)\}\}
\end{aligned}
$$

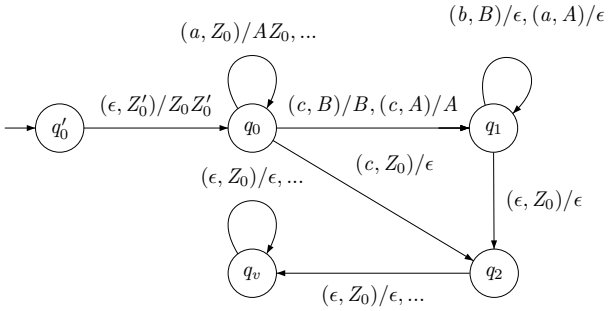

Figura 4.10: Autômato M' equivalente com critério de aceitação de pilha vazia

□

Teorema 4.12 (Estado final ⇐ pilha vazia) *Seja M um autômato de pilha não-determinístico com critério de aceitação baseado em pilha vazia. Então, é possível definir um autômato de pilha não-determinístico M' com critério de aceitação baseado em estado final, de modo que $V(M) = L(M')$.*

Justificativa O Algorimo 4.10 apresenta um método para se efetuar tal conversão.

Algoritmo 4.10 (Estado final ⇐ pilha vazia) *Obtenção de um autômato de pilha baseado em critério de aceitação de estado final a partir de outro baseado em critério de pilha vazia.*

- Entrada: um autômato de pilha não-determinístico $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, com critério de aceitação baseado em pilha vazia.
- Saída: um autômato de pilha não-determinístico $M' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', \varnothing)$, com critério de aceitação baseado em estado final.
- Método:

  1. $Q' \leftarrow Q \cup \{q_f, q_0'\}$ (supondo-se, sem perda de generalidade, que $Q \cap \{q_f, q_0'\} = \varnothing$);
  2. $\Gamma' \leftarrow \Gamma \cup \{Z_0'\}$ (supondo-se, sem perda de generalidade, que $\Gamma \cap \{Z_0'\} = \varnothing$);
  3. Função de transição δ':

     a) $\delta' \leftarrow \varnothing$;

     b) $\delta'(q_0', \epsilon, Z_0') \leftarrow \{(q_0, Z_0Z_0')\}$;

     c) $\delta'(q, \sigma, \phi) \leftarrow \delta(q, \sigma, \phi), \quad \forall\, q \in Q, \phi \in \Gamma, \sigma \in (\Sigma \cup \{\epsilon\})$;

     d) $\delta'(q, \epsilon, Z_0') \leftarrow \{(q_f, \epsilon)\}, \quad \forall\, q \in Q$.

Assim como no caso anterior, o objetivo é construir M' capaz de simular o comportamento de M. Além disso, sempre que M esvaziar a sua pilha, o objetivo é fazer com

que M' transite para um estado final. Dessa maneira, as sentenças que conduzem M a uma configuração final causarão o mesmo efeito sobre M'.

A fim de detectar o esvaziamento da pilha em M, M' exibe, inserido no fundo de sua pilha, logo ao início do reconhecimento, um novo símbolo de pilha Z_0'. A transição criada no passso (3.b) do algoritmo tem por objetivo inserir na pilha, acima de Z_0', o símbolo de pilha inicial de M, permitindo que as transições do passo (3.c) reproduzam o comportamento de M sobre M'.

Finalmente, estando M' em qualquer estado, a simples presença de Z_0' no topo da pilha é informação suficiente para caracterizar o esvaziamento da pilha em M, causando a transição de M' para o estado final q_f. Dessa forma, como se pode perceber intuitivamente, cada configuração final em M tem a sua configuração correspondente em M', e ambos os autômatos de pilha aceitam portanto a mesma linguagem. ■

Exemplo 4.32 Considere o autômato de pilha da Figura 4.11, cujo critério de aceitação é baseado em pilha vazia.

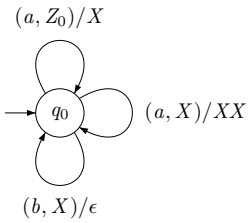

Figura 4.11: Autômato M com critério de aceitação de pilha vazia

A linguagem aceita por este autômato é constituída pelas sentenças sobre $\{a, b\}$ que contêm a mesma quantidade de símbolos a e b e, além disso, cuja quantidade acumulada de símbolos a seja sempre maior que a quantidade de símbolos b, contando-se os símbolos que formam a sentença, da esquerda para a direita. Exemplos de cadeias aceitas são $aaababbb$, ab e $abaababb$. Exemplos de cadeias não aceitas são a, ba, $aaabb$ e $abbbbaaa$.

Aplicando-se o Algoritmo 4.10 de conversão do critério de aceitação, obtém-se um novo autômato que reconhece a mesma linguagem e cujo critério é baseado em estado final (Figura 4.12).

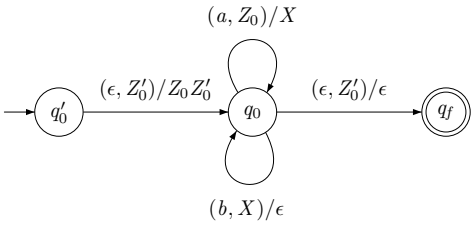

Figura 4.12: Autômato M' equivalente com critério de aceitação de estado final

□

4.8 Equivalência entre Gramáticas Livres de Contexto e Autômatos de Pilha

Apresentar-se-á aqui o resultado teórico que estabelece a equivalência da classe de linguagens aceita pelos autômatos de pilha não-determinísticos com a classe de linguagens geradas pelas gramáticas do tipo 2 — as linguagens livres de contexto gerais.

Essa apresentação é feita em duas partes. Inicialmente, mostra-se que para qualquer gramática livre de contexto é possível definir um autômato de pilha não-determinístico que reconhece exatamente a mesma linguagem gerada pela gramática. A seguir, é apresentado o resultado inverso, ou seja, de que toda e qualquer linguagem aceita por um autômato de pilha não-determinístico pode ser gerada por uma gramática livre de contexto.

Teorema 4.13 (Gramática ⇒ autômato de pilha) *Seja G uma gramática livre de contexto. Então é possível definir um autômato de pilha não-determinístico M, com critério de aceitação baseado em pilha vazia, de modo que $V(M) = L(G)$.*

Justificativa Considere-se $G = (V, \Sigma, P, S)$ na Forma Normal de Greibach, e $\epsilon \notin L(G)$. O Algoritmo 4.11 mostra como obter o autômato M a partir de G.

Algoritmo 4.11 (Gramática ⇒ autômato de pilha, versão 1) *Obtenção de um autômato de pilha não-determinístico a partir de uma gramática livre de contexto na Forma Normal de Greibach.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$ na Forma Normal de Greibach, e $\epsilon \notin L(G)$;
- Saída: um autômato de pilha $M = (Q, \Sigma, \Gamma, \delta, q, S, F)$, com critério de aceitação de pilha vazia, tal que $V(M) = L(G)$;
- Método:
  1. $Q \leftarrow \{q\}$;
  2. $\Gamma \leftarrow N$;
  3. $F \leftarrow \varnothing$;
  4. Função de transição:
     a) $\delta \leftarrow \varnothing$;
     b) $\delta(q, \sigma, A) = \{(q, \gamma) \mid A \rightarrow \sigma\gamma \in P\}, \forall\, A \in N, \sigma \in \Sigma, \gamma \in N^*$.

Note-se, no algoritmo acima, que se $\gamma = \epsilon$, então $\{(q, \epsilon)\} \subseteq \delta(q, \sigma, A)$.

É possível provar (ver [46]), por indução sobre a quantidade de movimentos realizados pelo autômato assim definido, que:

$$S \Rightarrow^* x$$

se e somente se

$$(q, x, S) \vdash^* (q, \epsilon, \epsilon)$$

Em outras palavras, se existe uma seqüência de derivações em G que gera uma cadeia x, então existe uma seqüência de movimentações em M que reconhece x, e vice-versa. Portanto, G e M definem a mesma linguagem. ■

Observe-se, no caso geral, que os autômatos construídos conforme o algoritmo 4.11 são não-determinísticos, uma vez que a simples presença de produções do tipo $A \rightarrow \sigma\gamma_1$ e $A \rightarrow \sigma\gamma_2$, em P, com $\gamma_1 \neq \gamma_2$, propicia a existência de mais de uma movimentação possível a partir da configuração $(q,\ \sigma z,\ A\beta)$:

$$(q, \sigma z, A\beta) \vdash (q, z, \gamma_1\beta) \quad \text{ou} \quad (q, \sigma z, A\beta) \vdash (q, z, \gamma_2\beta)$$

O Algoritmo 4.11 produz, como se pode perceber, um autômato de pilha que possui um único estado. A sua definição é idealizada com o intuito de permitir o reconhecimento de sentenças através de movimentos que simulem a seqüência de derivações mais à esquerda que seriam efetuadas pela gramática correspondente na geração das mesmas sentenças.

De acordo com essa definição, quando lido no sentido do topo para o fundo, o conteúdo da pilha é composto, em qualquer instante, por uma seqüência de símbolos não-terminais, tornados elementos do alfabeto de pilha do autômato. Mantidos nesta ordem, eles representam os símbolos não-terminais que ainda necessitariam sofrer substituição na correspondente forma sentencial que seria obtida através de derivações mais à esquerda na gramática fornecida. Por sua vez, as cadeias formadas pelos símbolos terminais que compõem o prefixo das formas sentenciais obtidas a partir de derivações mais à esquerda, em vez de serem mantidas na pilha, correspondem sempre à seqüência de símbolos que formam a porção já lida da fita de entrada.

Devido à própria organização em pilha (o primeiro elemento a entrar é o último a sair), o símbolo no topo da pilha representa sempre o próximo símbolo não-terminal a sofrer substituição em uma seqüência de derivações mais à esquerda, de acordo com a gramática correspondente. Como a gramática se encontra na Forma Normal de Greibach, todas as produções são do tipo $A \rightarrow \sigma\gamma$, com $\sigma \in \Sigma$ e $\gamma \in N^*$. Assim, a substituição do símbolo A, presente no topo da pilha, por γ ocorre apenas no caso em que o próximo símbolo na cadeia de entrada for σ. Nos casos em que isso não ocorre, não haverá tal possibilidade de movimentação, e a cadeia será rejeitada, pois a pilha não estará vazia. Ao avançar o cursor de leitura e modificar o conteúdo da pilha, o autômato de pilha simula exatamente a seqüência de movimentos que seriam executados pela aplicação da gramática correspondente através de derivações mais à esquerda.

Exemplo 4.33 Tome-se como exemplo a sentença $x \in L(G)$, e considere-se, em um dado momento da derivação mais à esquerda de x através de G, a forma sentencial $yA\beta$ com $y \in \Sigma$, $A \in N$ e $\beta \in N^*$. Seja $A \rightarrow \sigma\gamma$ a próxima produção a ser aplicada:

$$S \Rightarrow^* yA\beta \Rightarrow^* y\sigma\gamma\beta \Rightarrow^* y\sigma z$$

com $x = y\sigma z$.

Neste caso, o comportamento do autômato de pilha pode ser representado, em cada uma dessas quatro fases, através das figuras apresentadas a seguir, que ilustram as seguintes seqüências de movimentos:

$$(q, x, S) \vdash^* (q, \sigma z, A\beta) \vdash (q, z, \gamma\beta) \vdash^* (q, \epsilon, \epsilon)$$

Examinem-se, pois, as figuras que ilustram essas configurações. Na configuração inicial, a pilha contém apenas S e o cursor aponta para o primeiro símbolo da cadeia de entrada x:

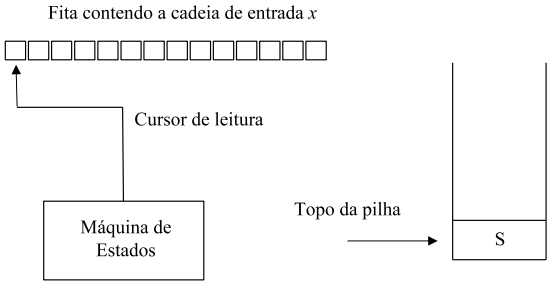

Figura 4.13: Autômato de pilha do Exemplo 4.33: configuração inicial

Após alguns movimentos, a parte já consumida da cadeia de entrada é y, o símbolo no topo da pilha é A, o restante da pilha é β, e o próximo símbolo na cadeia de entrada é σ:

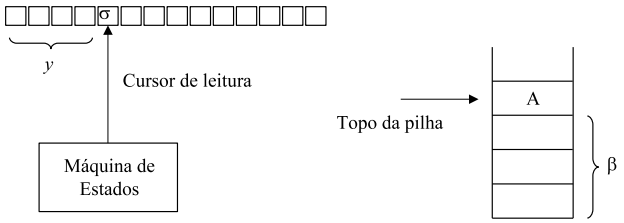

Figura 4.14: Autômato de pilha do Exemplo 4.33: antes da aplicação de uma regra

Imediatamente após a aplicação da produção $A \rightarrow \sigma\gamma$, a parte consumida da cadeia de entrada torna-se $y\sigma$ e o conteúdo da pilha é modificado para $\gamma\beta$:

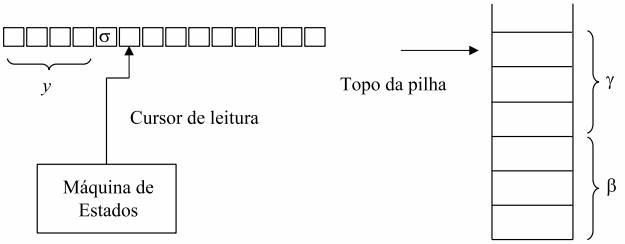

Figura 4.15: Autômato de pilha do Exemplo 4.33: depois da aplicação de uma regra

Finalmente, ao término da leitura da cadeia x, a parte consumida da cadeia de entrada torna-se exatamente x e a pilha é esvaziada por completo, caracterizando portanto a configuração de aceitação do autômato:

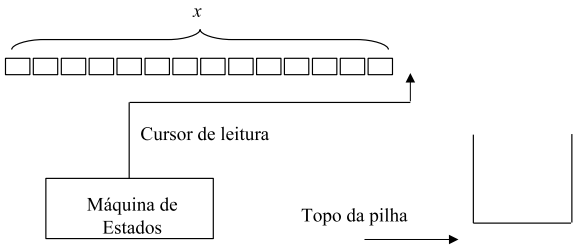

Figura 4.16: Autômato de pilha do Exemplo 4.33: configuração final

□

Exemplo 4.34 Considere-se a linguagem das expressões aritméticas na notação infixa, cuja gramática na Forma Normal de Greibach foi obtida no Exemplo 4.26. Os não-terminais T', E' e $)'$ foram renomeados para Y, Z e X, respectivamente:

$$\begin{aligned}
\{F &\rightarrow (EX \mid a,\\
T &\rightarrow (EX \mid a \mid (EXY \mid aY,\\
E &\rightarrow (EX \mid a \mid (EXY \mid aY \mid (EXZ \mid aZ \mid (EXYZ \mid aYZ,\\
Y &\rightarrow *F \mid *FY,\\
Z &\rightarrow +T \mid +TZ,\\
X &\rightarrow)\}
\end{aligned}$$

Aplicando-se o Algoritmo 4.11, obtém-se o autômato de pilha não-determinístico cuja função de transição é apresentada a seguir:

$$\begin{aligned}
\{(q,(,F) &\rightarrow \{(q,EX)\},\\
(q,a,F) &\rightarrow \{(q,\epsilon)\},\\
(q,(,T) &\rightarrow \{(q,EX),(q,EXY)\},\\
(q,a,T) &\rightarrow \{(q,\epsilon),(q,Y)\},\\
(q,(,E) &\rightarrow \{(q,EX),(q,EXY),(q,EXZ),(q,EXYZ)\},\\
(q,a,E) &\rightarrow \{(q,\epsilon),(q,Y),(q,Z),(q,YZ)\},\\
(q,*,Y) &\rightarrow \{(q,F),(q,FY)\},\\
(q,+,Z) &\rightarrow \{(q,T),(q,TZ)\},\\
(q,),X) &\rightarrow \{(q,\epsilon)\}\}
\end{aligned}$$

□

Cumpre notar, na demonstração informal do Teorema 4.13, que, como conseqüência da exigência de que G se apresente na Forma Normal de Greibach, com produções obedecendo exclusivamente ao formato $A \rightarrow \sigma\gamma$, com $\sigma \in \Sigma$, que esta demonstração não se aplica ao caso das linguagens livres de contexto que incluam a cadeia vazia. A fim de generalizar o resultado desse teorema, deve-se introduzir uma pequena modificação para contemplar o caso de linguagens livres de contexto que incluam a cadeia vazia.

A modificação em questão consiste na introdução de um novo estado q_0 em M, renomeando-se o estado "q" para "q_1". Duas novas transições devem também ser acrescentadas:

$$\delta(q_0, \epsilon, S) = \{(q_0, \epsilon), (q_1, S)\}$$

A transição $\delta(q_0, \epsilon, S) = \{(q_0, \epsilon)\}$ permite que o autômato remova S da pilha sem consumir símbolos da cadeia de entrada, possibilitando, portanto, o reconhecimento de ϵ.

A transição $\delta(q_0, \epsilon, S) = \{(q_1, S)\}$ preserva o conteúdo da pilha, conduzindo o autômato ao estado convencional utilizado para o reconhecimento de $L - \{\epsilon\}$.

Observe-se que a simples inclusão da transição $\delta(q, \epsilon, S) = \{(q, \epsilon)\}$ na máquina M anteriormente definida não é solução para o problema apresentado, uma vez que permite a remoção do símbolo S em outras configurações que não apenas a inicial, podendo causar alteração na linguagem L originalmente definida. Na verdade, a inclusão de um estado adicional simula a existência de uma nova raiz para a gramática-base, impedindo que a remoção de S ocorra em outras configurações que não a inicial, além de possibilitar o reconhecimento da cadeia vazia se esta pertencer à linguagem definida por G.

Exemplo 4.35 Considere-se a gramática G definida pelo seguinte conjunto de produções:

$$\begin{array}{rcl} \{S & \rightarrow & aSBS, \\ S & \rightarrow & aB, \\ B & \rightarrow & b\} \end{array}$$

A linguagem definida por esta gramática, $L(G)$, é aceita por um autômato M que tenha a seguinte função de transição:

$$\begin{array}{rcl} \delta(q, a, S) & = & \{(q, SBS), (q, B)\} \\ \delta(q, b, B) & = & \{(q, \epsilon)\} \end{array}$$

A menor e a segunda menor sentenças pertencentes a $L(G)$ são, respectivamente:

- $S \Rightarrow aB \Rightarrow ab$
- $S \Rightarrow aSBS \Rightarrow aaBBS \Rightarrow^* aabbS \Rightarrow^* aabbab$

Logo, $L(G) = \{ab, aabbab, \ldots\}$. Considerando-se a linguagem $L' = L \cup \{\epsilon\}$, pode-se imediatamente observar que a mera inclusão da produção $S \rightarrow \epsilon$ em G não produziria o efeito desejado, que é o de permitir a geração de L', uma vez que ela incluiria a geração, por exemplo, da cadeia $aabb$, originariamente não pertencente a L. Por esse mesmo motivo, a transição:

$$\delta(q, \epsilon, S) = \{(q, \epsilon)\}$$

não produziria o efeito desejado, que é o de permitir o reconhecimento de $L \cup \{\epsilon\}$ a partir do autômato-base M. Para isso, é preciso acrescentar o estado adicional, tornando a função δ de transição do autômato M:

$$\begin{array}{rcl} \delta(q_0, \epsilon, S) & = & \{(q_0, \epsilon), (q_1, S)\} \\ \delta(q_1, a, S) & = & \{(q_1, SBS), (q_1, B)\} \\ \delta(q_1, b, B) & = & \{(q_1, \epsilon)\} \end{array}$$

Note a correspondência do conjunto de produções assim definido com a gramática G' abaixo apresentada, $L' = L'(G')$:

$$\begin{array}{rcl} Z & \rightarrow & S \\ Z & \rightarrow & \epsilon \\ S & \rightarrow & aSBS \\ S & \rightarrow & aB \\ B & \rightarrow & b \end{array}$$

□

Naturalmente, dada uma gramática G em que S é a raiz, sendo que S garantidamente não comparece no lado direito de nenhuma produção, então torna-se desnecessário acrescentar um estado adicional, se o objetivo de sua inclusão for exclusivamente o de

permitir o reconhecimento da cadeia vazia em adição à linguagem original. Em casos particulares como esse, a transição:

$$\delta(q, \epsilon, S) = \{(q, \epsilon)\}$$

produz o efeito desejado, podendo ser incluída em M sem que haja risco de se alterar a linguagem definida pelo autômato.

Uma forma alternativa de se especificar um autômato de pilha M que reconheça exatamente a mesma linguagem definida por uma gramática livre de contexto qualquer, não necessariamente na Forma Normal de Greibach, e eventualmente incluindo a cadeia vazia, pode também ser obtida através do Algoritmo 4.12, a seguir apresentado.

Algoritmo 4.12 (Gramática $\Rightarrow$ autômato de pilha, versão 2) *Obtenção de um autômato de pilha não-determinístico a partir de uma gramática livre de contexto qualquer.*

- Entrada: uma gramática livre de contexto $G = (V, \Sigma, P, S)$;
- Saída: um autômato de pilha não-determinístico $M = (Q, \Sigma, \Gamma, q, S, F)$ com critério de aceitação de pilha vazia, tal que $V(M) = L(G)$;
- Método:
  1. $Q \leftarrow \{q\}$;
  2. $\Gamma \leftarrow V$;
  3. $F \leftarrow \varnothing$;
  4. Função de transição:
     a) $\delta \leftarrow \varnothing$;
     b) $\delta(q, \epsilon, A) = \{(q, \gamma) \mid A \rightarrow \gamma \in P\}, \forall\, A \in N, \gamma \in V^*$;
     c) $\delta(q, \sigma, \sigma) = \{(q, \epsilon)\}, \forall\, \sigma \in \Sigma$.

Observe-se, neste caso, que o alfabeto de pilha é o próprio vocabulário V da gramática, e não apenas o conjunto dos não-terminais N como no caso anterior. Autômatos construídos segundo este critério operam através da repetida substituição dos símbolos não-terminais no topo da pilha, sem consumo de símbolos da fita de entrada, até que surja um símbolo terminal do topo da pilha. Nesta configuração, a sua remoção é condicionada à presença do mesmo símbolo na posição de leitura correntemente apontada pelo cursor da fita de entrada.

Assim como no caso anterior, autômatos construídos segundo este critério também simulam a seqüência de derivações mais à esquerda que seria feita pela gramática correspondente na geração da mesma sentença. Note-se que:

1. Pelo primeiro método, a escolha da produção a ser aplicada depende do não-terminal no presente topo da pilha e também do próximo símbolo de entrada. Não-

determinismos, neste caso, ocorrem apenas quando há mais de uma alternativa de substituição para um mesmo símbolo não-terminal começando com o mesmo símbolo terminal:

- $A \rightarrow \sigma\gamma_1 \quad \text{e} \quad A \rightarrow \sigma\gamma_2 \quad \text{geram} \quad \delta(q, \sigma, A) = \{(q, \gamma_1), (q, \gamma_2)\}$

Por outro lado, regras iniciadas com símbolos distintos não geram não-determinismo:

- $A \rightarrow \sigma_1\gamma_1 \quad \text{gera} \quad \delta(q, \sigma_1, A) = \{(q, \sigma_1\gamma_1)\}$
- $A \rightarrow \sigma_2\gamma_2 \quad \text{gera} \quad \delta(q, \sigma_2, A) = \{(q, \sigma_2\gamma_2)\}$

2. Pelo segundo método, a escolha da produção a ser aplicada depende apenas do não-terminal presente no topo da pilha. Dessa forma, o conjunto de opções de substituição para um mesmo símbolo não-terminal é geralmente maior do que no primeiro caso:

   - $A \rightarrow \sigma\gamma_1 \quad \text{e} \quad A \rightarrow \sigma\gamma_2 \quad \text{geram} \quad \delta(q, \epsilon, A) = \{(q, \sigma\gamma_1), (q, \sigma\gamma_2)\}$

   Essa situação se repete caso as regras se iniciem com símbolos distintos:

   - $A \rightarrow \sigma_1\gamma_1 \quad \text{e} \quad A \rightarrow \sigma_2\gamma_2 \quad \text{geram} \quad \delta(q, \epsilon, A) = \{(q, \sigma_1\gamma_1), (q, \sigma_2\gamma_2)\}$

Constata-se, dessa maneira, que o segundo método gera uma situação de não-determinismo para regras que se iniciam com símbolos diferentes, o que não ocorre no primeiro método. Assim, o segundo método tende a apresentar mais situações não-determinísticas do que o primeiro, dando a este uma certa vantagem quanto à potencial eficiência do reconhecimento.

Irrelevante do ponto de vista teórico, a questão da escolha de um ou outro método para a obtenção de autômatos de pilha a partir de gramáticas livres de contexto apresenta um certo interesse quando se trata da realização física de reconhecedores sintáticos eficientes para linguagens livres de contexto quaisquer, uma vez que o primeiro método possibilita, no caso geral, a construção de autômatos não-determinísticos mais eficientes do que os obtidos através da aplicação do segundo método, em decorrência da limitação na quantidade de movimentos considerados válidos em cada configuração.

Na prática, no entanto, o maior interesse recai sobre os autômatos de pilha determinísticos. Porém, conforme será visto na Seção 4.11, a classe de linguagens por eles reconhecida é apenas um subconjunto das linguagens livres de contexto. Linguagens livres de contexto genéricas só podem ser reconhecidas através de autômatos de pilha não-determinísticos, e por esse motivo o primeiro método é mais interessante do ponto de vista de engenharia, quando se trata da realização física de tais reconhecedores.

Exemplo 4.36 Considere-se a gramática das expressões aritméticas utilizada no Exemplo 4.26, empregada originalmente como base para a obtenção da gramática equivalente na Forma Normal de Greibach. No conjunto de produções abaixo, os não-terminais E' e T' foram respectivamente renomeados para Z e Y.

$$\begin{aligned} \{E &\rightarrow T \mid TZ, \\ T &\rightarrow F \mid FY, \end{aligned}$$

$$\begin{aligned} F &\rightarrow (E) \mid a, \\ Y &\rightarrow *F \mid *FY, \\ Z &\rightarrow +T \mid +TZ\} \end{aligned}$$

Aplicando-se o método alternativo (Algoritmo 4.12), obtém-se o autômato de pilha não-determinístico cuja função de transição δ é:

$$\begin{aligned} \{(q,\epsilon,E) &\rightarrow \{(q,T),(q,TZ)\}, \\ (q,\epsilon,T) &\rightarrow \{(q,F),(q,FY)\}, \\ (q,\epsilon,F) &\rightarrow \{(q,(E)),(q,a)\}, \\ (q,\epsilon,Y) &\rightarrow \{(q,*F),(q,*FY)\}, \\ (q,\epsilon,Z) &\rightarrow \{(q,+T),(q,+TZ)\}, \\ (q,a,a) &\rightarrow \{(q,\epsilon)\}, \\ (q,(,() &\rightarrow \{(q,\epsilon)\}, \\ (q,),)) &\rightarrow \{(q,\epsilon)\}, \\ (q,+,+) &\rightarrow \{(q,\epsilon)\}, \\ (q,*,*) &\rightarrow \{(q,\epsilon)\}\} \end{aligned}$$

□

Será examinada a seguir a equivalência entre os autômatos de pilha e as gramáticas do tipo 2, no que se refere à classe de linguagens que ambos os dispositivos são capazes de definir.

Teorema 4.14 (Gramática $\Leftarrow$ autômato de pilha) *Seja M um autômato de pilha com critério de aceitação de pilha vazia. Então é possível definir uma gramática livre de contexto G, de modo que $L(G) = V(M)$.*

Justificativa Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. A gramática $G = (V, \Sigma, P, S)$, tal que $L(G) = V(M)$, pode ser obtida pela aplicação do Algoritmo 4.13.

Algoritmo 4.13 (Gramática $\Leftarrow$ autômato de pilha) *Obtenção de uma gramática livre de contexto a partir de um autômato de pilha.*

- Entrada: um autômato de pilha $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ com critério de aceitação de pilha vazia;
- Saída: uma gramática livre de contexto $G = (V, \Sigma, P, S)$, tal que $L(G) = V(M)$;
- Método:

  1. $N \leftarrow \{[q_i Z q_k] \mid q_i, q_k \in Q \text{ e } Z \in \Gamma\} \cup \{S\}$;
  2. $P \leftarrow \varnothing$;
  3. $\forall\, q, q_1 \in Q, \sigma \in \Sigma \cup \{\epsilon\}, Z, Z_1, \ldots, Z_k \in \Gamma$, se $\{(q_1, Z_1 \ldots Z_k)\} \subseteq \delta(q, \sigma, Z)$, $k \geq 0$, então:

     $P \leftarrow P \cup \{[qZq_{k+1}] \rightarrow \sigma[q_1 Z_1 q_2][q_2 Z_2 q_3] \ldots [q_k Z_k q_{k+1}]\}$

     para qualquer seqüência de $q_j \in Q$, $2 \leq j \leq (k+1)$;

4. $P \leftarrow P \cup \{S \rightarrow [q_0 Z_0 q]\}, \forall\, q \in Q.$

É possível demonstrar formalmente (ver [46]), por indução sobre a quantidade de derivações efetuadas através de G, que a gramática assim definida gera exatamente a mesma linguagem aceita pelo autômato, ou seja:

$$(q_0, x, Z_0) \vdash^* (q, \epsilon, \epsilon) \quad \text{se e somente se} \quad S \Rightarrow^* x$$

Para que se possa intuir o significado do método apresentado, deve-se inicialmente notar que os símbolos não-terminais $[q_i Z q_k]$ criados na construção de G têm por objetivo representar o conjunto das sentenças α que são aceitas entre as duas configurações sugeridas pelo nome atribuído ao não-terminal:

$$(q_i, \alpha\beta, Z_0) \vdash^* (q_k, \beta, \mu)$$

Assim, q_i e Z devem ser interpretados como uma determinada configuração do autômato, e q_k o estado que ele atinge após a realização de uma série de movimentos, ao término dos quais o símbolo Z é removido da pilha. O conjunto das cadeias α que permitem ao autômato transitar de uma para outra configuração constitui uma linguagem, podendo ser representada como:

$$L([q_i Z q_k]) = \{\alpha \in \Sigma^* \mid [q_i Z q_k] \rightarrow \alpha\}$$

Conseqüentemente, como a linguagem aceita pelo autômato é formada pelo conjunto das cadeias que o levam desde a configuração inicial única até uma configuração final, não necessariamente única, tem-se que esta linguagem pode ser formalizada, do ponto de vista gramatical, como:

$$L(G) = \bigcup_{q_i \in Q} L([q_0 Z_0 q_i])$$

Observe-se que derivações mais à esquerda em G simulam os movimentos de M ao longo de suas diversas configurações. Para justificar a necessidade de incorporação da informação de estado à definição dos não-terminais de G, considere-se inicialmente $N = \Gamma$, ou seja, suponha-se que os não-terminais de G correspondem aos elementos do alfabeto de pilha de M e, além disso, que as produções de G sejam definidas de maneira simplificada, com base exclusivamente no alfabeto de pilha e nas diversas substituições possíveis para elementos no topo da pilha, dispensando-se a informação dos estados em sua definição.

Essa simplificação possibilitaria a aplicação incondicional de produções do tipo $A \rightarrow \sigma\gamma$ em qualquer forma sentencial, obtida a partir de G, em que A estivesse presente.

Claramente, não é essa a informação que o autômato traz, uma vez que a substituição de A por $\sigma\gamma$ é possível apenas em algumas situações ou estados: aqueles para os quais $\{(r, \gamma)\} \subseteq \delta(q, \sigma, A)$. Dessa forma, a caracterização de um não-terminal deveria necessariamente levar em conta a informação do estado (ou grupo de estados) no qual a substituição em questão pode ser feita. Caso contrário, corre-se o risco de alterar a linguagem definida pela gramática, tornando-a diferente da linguagem aceita pelo autômato em questão.

Em outras palavras, note-se que a presença de um não-terminal A em uma forma sentencial gerada a partir de derivações mais à esquerda em G:

$$S \Rightarrow^* \alpha A \beta, \quad \text{com } \alpha \in \Sigma^*, \beta \in V^*$$

deveria corresponder, em M, ao caso em que A é o símbolo presente no topo da pilha e σ, o primeiro símbolo terminal derivável a partir de $A\beta$, ou seja, $A\beta \stackrel{*}{\Rightarrow} \sigma\,\mu$. A substituição de A por $\sigma\gamma$ nessa forma sentencial, para uma produção $A \rightarrow \sigma\gamma$ da gramática, deve ser admitida como alternativa de substituição válida se e apenas se a transição:

$$\{(q_j, \gamma)\} \subseteq \delta(q_i, \sigma, A)$$

constituir alternativa de movimentação a partir da configuração $(q_i, \sigma\,\mu, A)$:

$$(q_0, \alpha\sigma, Z_0) \vdash^* (q_i, \sigma, A\nu) \vdash (q_j, \mu, \gamma\,\mu)$$

Caso a informação de estado não esteja incorporada à definição dos não-terminais de G, conforme discutido, corre-se o risco de admitir a aplicação das produções de G em formas sentenciais sem qualquer correspondência com as configurações de M. Justamente para evitar essa situação, e visando estabelecer a plena equivalência entre as formas sentenciais de G e as configurações de M, é que a informação de estado deve ser agregada a cada símbolo $A \in N$, conforme originalmente proposto.

Assim, definem-se os símbolos não-terminais de G como:

$$[q_i A r], \quad \text{com } q_i, r \in Q, A \in \Gamma$$

Supondo-se que $\{(r, \gamma)\} \subseteq \delta(q_i, \sigma, A)$, garante-se que a produção $A \rightarrow \sigma\gamma$ seja empregada apenas naquelas formas sentenciais geradas por G em que o não-terminal $[q_i A r]$ estiver presente, não se manifestando nunca como alternativa de substituição para os demais não-terminais $[q_k A r]$, $k \neq i$, caso $\{(r, \gamma)\}$ não esteja contido em $\delta(q_k, \sigma, A)$.

Tendo-se analisado o significado atribuído aos não-terminais definidos para G, a interpretação dos critérios de construção do conjunto de produções torna-se mais fácil: cada produção de G representa a evolução do correspondente autômato ao longo de uma série de configurações, série esta que se inicia com o consumo de um símbolo da cadeia de entrada (ou nenhum, se a transição for em vazio), e termina com a remoção de todos os símbolos armazenados na pilha como conseqüência da aplicação dessa transição. Dessa forma, as produções definidas para S representam todas as possíveis seqüências de movimentações que levam M da configuração inicial até uma de suas várias possíveis configurações finais — adotando-se o critério de aceitação por esvaziamento da pilha. ■

Como efeito colateral da aplicação do método apresentado, é freqüente a obtenção de gramáticas com elevada quantidade de símbolos não-terminais e produções inúteis. Apesar de a aplicação dos métodos anteriormente estudados possibilitar a eliminação sistemática de tais elementos inúteis, é possível limitar a sua proliferação desde o início, restringindo-se a aplicação do Algoritmo 4.13 apenas aos novos símbolos não-terminais que surgem em cada passo.

Exemplo 4.37 Seja $M = (\{q_0, q_1\}, \{a, b\}, \{X, Z_0\}, \delta, q_0, Z_0, \varnothing)$, representado na Figura 4.17.

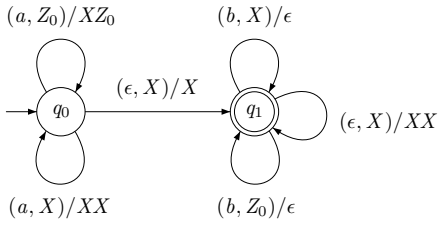

Figura 4.17: Autômato de pilha M do Exemplo 4.37

A obtenção de G, tal que $L(G) = V(M)$, tem como ponto de partida as duas produções do conjunto:

$$\begin{aligned} \{S &\rightarrow [q_0 Z_0 q_0], && (4.7)\\ S &\rightarrow [q_0 Z_0 q_1]\} && (4.8) \end{aligned}$$

Considerando-se inicialmente (4.7), definem-se as produções de G que apresentam $[q_0 Z_0 q_0]$ do lado esquerdo. Essas produções decorrem das transições de M que emergem do estado q_0 com o símbolo do alfabeto de pilha Z_0. Como há apenas uma transição que satisfaz a esse requisito,

$$\delta(q_0, a, Z_0) = (q_0, XZ_0)$$

obtém-se por conseguinte o conjunto de produções:

$$\begin{aligned} \{[q_0 Z_0 q_0] &\rightarrow a[q_0 X q_0][q_0 Z_0 q_0], && (4.9)\\ [q_0 Z_0 q_0] &\rightarrow a[q_0 X q_1][q_1 Z_0 q_0]\} && (4.10) \end{aligned}$$

Como se pode observar, (4.10) contém o símbolo $[q_1 Z_0 q_0]$, que, de acordo com o que foi convencionado, representa o conjunto de cadeias que conduzem M do estado q_1 ao estado q_0 com o símbolo Z_0. Através da inspeção do diagrama de estados de M, conclui-se não haver caminhos com esta característica, e por esse motivo antecipa-se a impossibilidade de se criarem produções para o não-terminal $[q_1 Z_0 q_0]$. Em outras palavras, a definição de uma produção para esse não-terminal dependeria, antes de mais nada, da existência de uma transição que, no estado q_1, estivesse associada ao símbolo Z_0, o que não é verdade para o autômato considerado. Desse modo, descarta-se a produção (4.10). Como conseqüência, também a produção (4.9) torna-se inútil, pois não há termo independente de $[q_0 Z_0 q_0]$ que possibilite a resolução da equação representada pela produção.

Prosseguindo com a produção (4.8), considere-se o símbolo $[q_0 Z_0 q_1]$. As produções a ele associadas também derivam da transição que vincula o símbolo Z_0 ao estado q_0:

$$\delta(q_0, a, Z_0) = (q_0, XZ_0)$$

$$\begin{aligned} \{[q_0 Z_0 q_1] &\rightarrow a[q_0 X q_0][q_0 Z_0 q_1], && (4.11)\\ [q_0 Z_0 q_1] &\rightarrow a[q_0 X q_1][q_1 Z_0 q_1]\} && (4.12) \end{aligned}$$

A seguir, obtém-se para o símbolo $[q_0 X q_0]$, presente na produção (4.11), o seguinte conjunto de produções, derivado da transição:

$$\delta(q_0, a, X) = (q_0, XX)$$

$$\begin{aligned} \{[q_0 X q_0] &\rightarrow a[q_0 X q_0][q_0 X q_0], && (4.13)\\ [q_0 X q_0] &\rightarrow a[q_0 X q_1][q_1 X q_0]\} && (4.14) \end{aligned}$$

Por motivos similares aos expostos para o caso da produção (4.10), conclui-se que não é possível definir produções para o símbolo $[q_1 X q_0]$, o que implica, como conseqüência, a inutilidade do símbolo $[q_0 X q_0]$ e das produções (4.14), (4.13) e (4.11).

Prosseguindo com a produção (4.12), definem-se as produções para o símbolo $[q_0 X q_1]$. Nesse caso, elas decorrem das transições:

$$\delta(q_0, a, X) = (q_0, XX) \quad \text{e} \quad \delta(q_0, \epsilon, X) = (q_1, X)$$

$$\begin{aligned} \{[q_0 X q_1] &\rightarrow a[q_0 X q_0][q_0 X q_1], && (4.15)\\ [q_0 X q_1] &\rightarrow a[q_0 X q_1][q_1 X q_1], && (4.16)\\ [q_0 X q_1] &\rightarrow [q_1 X q_1]\} && (4.17) \end{aligned}$$

Observe-se que as duas primeiras produções são decorrência da primeira transição, ao passo que a terceira provém da segunda transição. Como $[q_0 X q_0]$ é um símbolo inútil, isso acarreta

a inutilidade da produção (4.15). Continuando com a produção (4.12), define-se agora a única produção do símbolo $[q_1 Z_0 q_1]$, e que deriva da transição:

$$\delta(q_1, b, Z_0) = (q_1, \epsilon)$$

$$[q_1 Z_0 q_1] \quad \rightarrow \quad b \tag{4.18}$$

Finalmente, para completar a definição dos símbolos não-terminais até aqui surgidos, resta obter as produções de $[q_1 X q_1]$, utilizado nas produções (4.16) e (4.17). Note-se que estas decorrem, respectivamente, das transições:

$$\delta(q_1, b, X) = (q_1, \epsilon) \quad \text{e} \quad \delta(q_1, \epsilon, X) = (q_1, XX)$$

$$\{[q_1 X q_1] \quad \rightarrow \quad b, \tag{4.19}$$
$$[q_1 X q_1] \quad \rightarrow \quad [q_1 X q_1][q_1 X q_1]\} \tag{4.20}$$

Como resultado, obtém-se o seguinte conjunto de produções:

$$\begin{array}{ll}
(4.8) & \{S \rightarrow [q_0 Z_0 q_1], \\
(4.12) & [q_0 Z_0 q_1] \rightarrow a[q_0 X q_1][q_1 Z_0 q_1], \\
(4.16) & [q_0 X q_1] \rightarrow a[q_0 X q_1][q_1 X q_1], \\
(4.17) & [q_0 X q_1] \rightarrow [q_1 X q_1], \\
(4.18) & [q_1 Z_0 q_1] \rightarrow b, \\
(4.19) & [q_1 X q_1] \rightarrow b, \\
(4.20) & [q_1 X q_1] \rightarrow [q_1 X q_1][q_1 X q_1]\}
\end{array}$$

Renomeando-se os não-terminais, obtém-se:

$$\begin{array}{ll}
(4.8) & \{S \rightarrow A, \\
(4.12) & A \rightarrow aBC, \\
(4.16) & B \rightarrow aBD, \\
(4.17) & B \rightarrow D, \\
(4.18) & C \rightarrow b, \\
(4.19) & D \rightarrow b, \\
(4.20) & D \rightarrow DD\}
\end{array}$$

ou seja, $L(G) = V(M) = \{a^i b^j \mid i \geq 1 \text{ e } j > i\}$.

Apresenta-se agora uma particular seqüência de movimentos efetuada por M durante o reconhecimento da sentença $aabbbb$:

$(q_0, aabbbb, Z_0) \vdash (q_0, abbbb, XZ_0) \vdash (q_0, bbbb, XXZ_0) \vdash (q_1, bbbb, XXZ_0) \vdash (q_1, bbb, XZ_0) \vdash (q_1, bbb, XXZ_0) \vdash (q_1, bb, XZ_0) \vdash (q_1, b, Z_0) \vdash (q_1, \epsilon, \epsilon)$

A correspondência entre G e M pode ser ilustrada através da análise da seqüência de derivações mais à esquerda obtida para esta mesma cadeia $aabbbb$, e de sua comparação com a seqüência de movimentos efetuados pelo autômato conforme apresentado acima:

$$\begin{array}{rl}
S \quad \Rightarrow & [q_0 Z_0 q_1] \Rightarrow a[q_0 X q_1][q_1 Z_0 q_1] \Rightarrow aa[q_0 X q_1][q_1 X q_1][q_1 Z_0 q_1] \Rightarrow \\
& aa[q_1 X q_1][q_1 X q_1][q_1 Z_0 q_1] \Rightarrow aab[q_1 X q_1][q_1 Z_0 q_1] \Rightarrow \\
& aab[q_1 X q_1][q_1 X q_1][q_1 Z_0 q_1] \Rightarrow \\
& aabb[q_1 X q_1][q_1 Z_0 q_1] \Rightarrow \\
& aabbb[q_1 Z_0 q_1] \Rightarrow \\
& aabbbb
\end{array}$$

Observe-se que as formas sentenciais obtidas através de G, neste exemplo, estão diretamente relacionadas às configurações assumidas por G em cada etapa do reconhecimento da cadeia considerada. Note-se, portanto, que G "simula", através de uma seqüência de derivações mais à esquerda, a seqüência de movimentos que conduz M de sua configuração inicial até uma configuração final.

Retornando à questão da caracterização dos não-terminais em G, conforme mencionado anteriormente, a incorporação dos estados na definição dos mesmos é necessária para evitar a geração de formas sentenciais inválidas, sendo, portanto, indispensável para garantir a equivalência entre a linguagem aceita e a linguagem gerada.

Este ponto pode ser melhor visualizado neste exemplo da seguinte forma: considere-se G' em que $N = \Gamma$ e P é obtido simplesmente analisando-se todas as possíveis substituições de símbolos no topo da pilha e a eventual presença de elementos de Σ na fita de entrada, sem levar em conta a existência dos estados.

Assim, obtém-se o conjunto de produções seguinte:

$$
\begin{aligned}
&\{S \to Z_0, && \\
&Z_0 \to aXX, && (\text{de } \delta(q_0, a, Z_0) = (q_0, XZ_0)) \\
&X \to aXX, && (\text{de } \delta(q_0, a, X) = (q_0, XX)) \\
&X \to X, && (\text{de } \delta(q_0, \epsilon, X) = (q_1, X)) \\
&X \to b, && (\text{de } \delta(q_1, b, X) = (q_1, \epsilon)) \\
&X \to XX, && (\text{de } \delta(q1, \epsilon, X) = (q_1, XX)) \\
&Z_0 \to b\} && (\text{de } \delta(q_1, b, Z_0) = (q_1, \epsilon))
\end{aligned}
$$

É fácil notar, neste caso, que $L(G) \subset L(G')$. Considere-se, a título de exemplo, a cadeia b. Evidentemente, $b \notin L(G)$, mas no entanto, em G', tem-se:

$$S \Rightarrow Z_0 \Rightarrow b$$

A razão disso acontecer é que a produção $Z_0 \to b$ é convertida para uma transição de M apenas no estado q_1. Ao ignorar a informação de estado na caracterização dos não-terminais de G', a substituição torna-se possível independentemente do estado. Neste exemplo, a derivação $Z_0 \Rightarrow b$ corresponderia à aplicação de uma transição originalmente inexistente no estado q_0. Deixa, portanto, de haver correspondência entre as configurações do reconhecedor e as formas sentenciais da gramática, quando a informação de estado é descartada na caracterização de seus símbolos não-terminais. □

4.9 Relação entre Linguagens Livres de Contexto e Linguagens Regulares

As linguagens regulares constituem um subconjunto próprio das linguagens livres de contexto. De fato, é possível demonstrar que toda linguagem regular é também uma linguagem livre de contexto e, por outro lado, que existem linguagens livres de contexto que não são regulares. Esses resultados serão demonstrados, respectivamente, nos Teoremas 4.15 e 4.16 a seguir.

Teorema 4.15 (Regulares $\subseteq$ livres de contexto) *Toda linguagem regular é também uma linguagem livre de contexto.*

Justificativa Considere-se L uma linguagem regular qualquer. Então, por definição, existe uma gramática linear à direita G que define L. As regras de G possuem todas, sem exceção, apenas um símbolo não-terminal do lado esquerdo e uma cadeia qualquer pertencente a $(\Sigma \cup \{\epsilon\})(N \cup \{\epsilon\})$ do lado direito. Como as gramáticas livres de contexto também exigem um único símbolo não-terminal no lado esquerdo das regras, e, além disso, $(\Sigma \cup \{\epsilon\})(N \cup \{\epsilon\}) \subset V^*NV^*$, isso implica que toda gramática linear à direita é também uma gramática livre de contexto. Logo, toda linguagem regular é também uma linguagem livre de contexto. ■

Teorema 4.16 (Regulares ≠ livres de contexto) *Existem linguagens livres de contexto que não são regulares.*

Justificativa Considere-se, por exemplo, a linguagem $L = \{0^i 10^i \mid i \geq 1\}$. Conforme demonstrado no Exemplo 3.48, L não é regular, uma vez que não satisfaz ao "Pumping Lemma" das linguagens regulares. No entanto, a gramática $G = (\{S, 0, 1\}, \{0, 1\}, \{S \rightarrow 0S0, S \rightarrow 010\}, S)$ gera exatamente L. Logo, L é livre de contexto, porém não é regular. De maneira semelhante, pode-se demonstrar a existência de inúmeras outras linguagens com essa mesma característica. ■

Conforme definido na Seção 4.1, uma linguagem é dita estritamente livre de contexto se ela for livre de contexto, porém não-regular. A característica desse tipo de linguagens é que elas são geradas apenas por gramáticas que possuam pelo menos um símbolo não-terminal que seja auto-recursivo central e essencial. Assim, é possível gerar, nas formas sentenciais geradas pela gramática, subcadeias da forma:

$$Y \Rightarrow^* \alpha Y \beta, \quad \text{com } \alpha, \beta \in \Sigma^+$$

as quais são responsáveis pelo balanceamento dos termos α e β nas sentenças da linguagem.

De fato, gramáticas lineares à direita não permitem a definição de símbolos não-terminais com essa propriedade. Ao contrário, gramáticas lineares à direita geram apenas formas sentenciais da seguinte forma:

$$Y \Rightarrow^* \alpha Y, \quad \text{com } \alpha \in \Sigma^+$$

em que, obviamente, não há balanceamento de termos nem, portanto, aninhamentos sintáticos. A existência de termos balanceados (ou aninhados) é, por isso, o fator que diferencia uma linguagem estritamente livre de contexto de uma linguagem regular.

4.10 Linguagens que não são Livres de Contexto

Como se sabe, as linguagens regulares podem ser definidas através de expressões regulares, gramáticas regulares ou autômatos finitos. Por outro lado, a existência de linguagens não-regulares foi provada através do uso do "Pumping Lemma" das linguagens regulares.

De maneira análoga, para provar que uma linguagem é livre de contexto, é suficiente apresentar uma gramática livre de contexto que gere esta linguagem, ou ainda um autômato de pilha que a reconheça. A existência de linguagens que não são livres de contexto pode ser demonstrada com o auxílio do "Pumping Lemma" das linguagens livres de contexto, apresentado a seguir.

A importância de um "Pumping Lemma" para as linguagens livres de contexto não se limita à posssibilidade de demonstrar a existência de linguagens que não sejam livres de contexto, e, portanto, de sugerir a existência de outras classes de linguagens. Ele é usado também na demonstração de algumas propriedades importantes das linguagens livres de contexto.

Teorema 4.17 ("Pumping Lemma" para linguagens livres de contexto) *Seja L uma linguagem livre de contexto, com $\epsilon \notin L$. Então, existe uma constante inteira n, dependente apenas de L, que satisfaz às seguintes condições: (i) $\forall \gamma \in L, |\gamma| \geq n, \gamma = uvwxy$; ; (ii) $|vwx| \leq n$; (iii) $|vx| \geq 1$; (iv) $\forall i \geq 0, uv^i wx^i y \in L$.*

Justificativa Se L é livre de contexto e não contém a cadeia vazia, então $L = L(G)$, sendo $G = (V, \Sigma, P, S)$, G uma gramática livre de contexto na Forma Normal de Chomsky. Portanto, qualquer árvore de derivação que represente uma sentença de L nessa gramática será uma árvore binária.

Por outro lado, árvores binárias de altura i geram sentenças de comprimento máximo 2^{i-1}. A Figura 4.18 ilustra essa relação para os casos i=1, 2 e 3, e facilita a generalização da mesma:

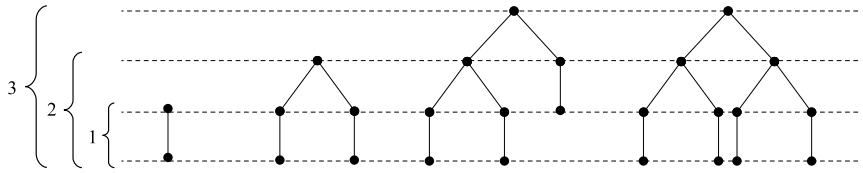

Figura 4.18: Comprimento máximo de um caminho em uma árvore binária

Através desta figura, é fácil perceber que árvores de altura 1 geram sentenças de comprimento 1 (portanto $\leq 2^0$), árvores de altura 2 geram sentenças de comprimento 2 ($\leq 2^1$) e árvores de altura 3 geram sentenças de comprimento 3 ou 4 ($\leq 2^2$). No caso geral, árvores com altura i geram sentenças de comprimento menor ou igual a 2^{i-1}.

Como conseqüência desse resultado, é possível inferir que, se uma sentença tem comprimento mínimo 2^i, então a árvore de derivação correspondente possuirá altura mínima $i + 1$.

Considere-se $k = |N|$, onde N é o conjunto dos símbolos não-terminais de G, estando esta expressa na Forma Normal de Chomsky, e faça-se $n = 2^k$.

Se $\gamma \in L$ e $|\gamma| \geq n$, isto é, se $|\gamma| \geq 2^k$, então, face ao resultado anterior, é certo que a altura mínima da árvore de derivação correspondente à sentença γ será $k + 1$. Suponha-se que o valor desta altura seja p (portanto, $p \geq k + 1$) e considere-se um caminho w qualquer na árvore que possua comprimento p (haverá pelo menos um caminho que satisfaça a essa condição).

Se o caminho selecionado possui comprimento p, então este caminho é formado de q símbolos, $q \geq k + 2$. Desses q símbolos, apenas um será terminal (o último símbolo do caminho, aquele que é folha da árvore) e os demais serão necessariamente não-terminais.

Ignorando o símbolo terminal e concentrando a atenção nos $q - 1$ não-terminais, seus antecessores, se $q \geq k + 2$, então existem r símbolos não-terminais neste caminho, e $r = q - 1$, ou seja, $r \geq k + 1$.

Considerem-se agora apenas os primeiros $k+1$ símbolos não-terminais consecutivos que antecedem imediatamente a folha da árvore, ignorando os $r - (k + 1)$ símbolos não-terminais situados no início do caminho selecionado. O caminho escolhido w pode, portanto, ser considerado como:

$$w = \mu\,\rho\sigma, \mu \in N^*, |\,\mu\,| = r - (k + 1), \rho \in N^+, |\rho| = k + 1, \sigma \in \Sigma$$

A Figura 4.19 ilustra esta interpretação do caminho w:

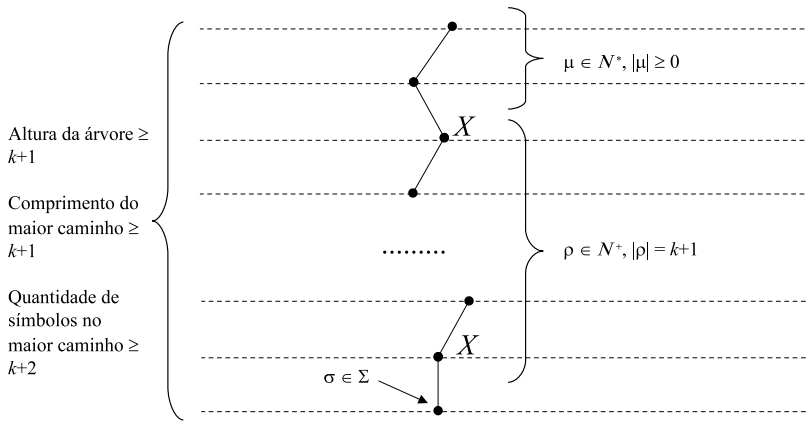

Figura 4.19: Ocorrência múltipla do símbolo não-terminal X no caminho w

Se ρ contém exatamente $k+1$ símbolos não-terminais e a gramática contém apenas k símbolos não-terminais distintos, é certo que pelo menos um símbolo não-terminal de G aparece mais de uma vez em ρ. Se chamarmos a este símbolo X, as seguintes considerações poderão ser feitas acerca das posições em que os símbolos X podem aparecer na árvore de derivação:

1. Para o *primeiro* X (aquele que está na posição mais alta da árvore, próxima da raiz):

   - Ele pode ser o primeiro símbolo da cadeia ρ (portanto, estar na altura $k+1$);
   - Ele pode ser o penúltimo símbolo da cadeia ρ (portanto, estar na altura 2), uma vez que o segundo X comparece por último nesta mesma cadeia;
   - Ele pode assumir qualquer posição entre essas duas.

2. Para o *segundo* X (aquele que está na posição mais baixa da árvore, próxima da folha):

   - Ele pode ser o segundo símbolo da cadeia ρ (portanto, estar na altura k), uma vez que o primeiro X comparece antes nesta mesma cadeia;
   - Ele pode ser o último símbolo da cadeia ρ (portanto, na altura 1);
   - Ele pode assumir qualquer posição entre essas duas.

Assim, pode-se concluir:

1. Para o *primeiro* X: se $X \Rightarrow^* \alpha_1, \alpha_1 \in \Sigma^*$, então $2 \leq |\alpha_1| \leq 2^k$;
2. Para o *segundo* X: se $X \Rightarrow^* \alpha_2, \alpha_2 \in \Sigma^*$, então $1 \leq |\alpha_2| \leq 2^{k-1}$.

A situação da árvore de derivação da sentença γ pode ser representada como mostra a Figura 4.20 (S representa a raiz de G, e pode, eventualmente, coincidir com o primeiro símbolo da cadeia ρ):

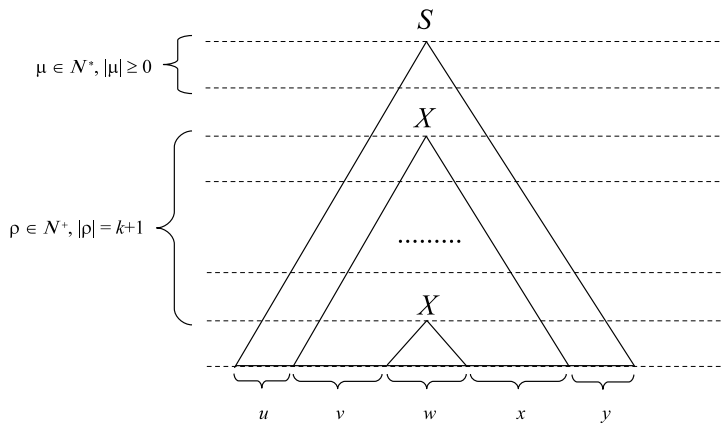

Figura 4.20: Árvore de derivação da cadeia $\gamma = uvwxy$

A inspeção desta árvore nos permite chegar às seguintes conclusões:

- A sentença γ pode ser considerada como sendo composta por cinco partes, $\gamma = uvwxy$;
- Como $w = \alpha_2$, então $1 \leq |w| \leq 2^{k-1}$;
- Como $vwx = \alpha_2$, então $2 \leq |vwx| \leq 2^k$;
- Como $1 \leq |w|$ e $2 \leq |vwx|$, então $|vx| \geq 1$.

Além disso, as regras de G permitem a derivação das seguintes formas sentenciais:

- $S \Rightarrow^* uXy$
- $X \Rightarrow^* vXx$
- $X \Rightarrow^* w$

A inspeção dessas formas sentenciais, assim como a análise da árvore mostrada na Figura 4.20, permite concluir que existem outras possibilidades de derivações (ou seja, de construção da árvore) de sentenças em G, as quais produzem sentenças diversas da original γ, e que, por construção, devem necessariamente pertencer à linguagem L.

Por exemplo, é possível imaginar que, em vez de aplicar regras que fazem o primeiro X derivar vXx, é possível aplicar, a esta ocorrência de X, as regras aplicadas ao segundo X, e dessa maneira gerar a sentença uwy no lugar da sentença $uvwxy$. Da mesma forma, seria possível repetir a derivação aplicada ao primeiro X no lugar da derivação feita para o segundo X, e com isso gerar a sentença $uvvwxxy$. Generalizando, todas as sentenças uv^iwx^iy, com $i \geq 0$, podem ser derivadas em G, e, portanto, pertencem necessariamente a $L(G)$. A Figura 4.21 mostra, graficamente, a aplicação desta idéia:

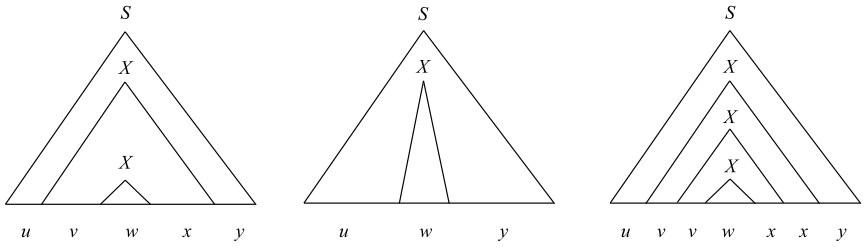

Figura 4.21: Árvores de derivação para as cadeias $uvwxy$, uwy e $uvvwxxy$

Em termos gramaticais, como $S \Rightarrow^* uXy$, $X \Rightarrow^* vXx$ e $X \Rightarrow^* w$, ou seja, $S \Rightarrow^* uvwxy$, então, necessariamente:

- $S \Rightarrow^* uXy \Rightarrow^* uwy$
- $S \Rightarrow^* uXy \Rightarrow^* uvvwxxy$
- $S \Rightarrow^* uXy \Rightarrow^* uvvvwxxxy$
- ...
- $S \Rightarrow^* uXy \Rightarrow^* uv^i wx^i y, \forall\, i \geq 0$

■

Considerado de maneira informal, este "Pumping Lemma" para linguagens livres de contexto pode ser interpretado da seguinte maneira:

- Sentenças que apresentem um certo comprimento mínimo geram àrvores de derivação com uma certa altura mínima;
- Como a quantidade de símbolos não-terminais da gramática é limitada, necessariamente deverá haver repetição de símbolos em algum caminho da árvore de derivação correspondente;
- Tal repetição permite a geração de uma quantidade infinita de outras sentenças estruturalmente similares, que devem necessariamente pertencer à linguagem.

O símbolo não-terminal X, na demonstração acima, possibilita derivações do tipo $X \Rightarrow^* vXx$ e corresponde ao símbolo que denominamos previamente de auto-recursivo central essencial. A existência de pelo menos um símbolo desse tipo na gramática garante a possibilidade de se gerar uma quantidade infinita de outras sentenças que também pertençam à mesma linguagem.

Se v ou x forem vazios, a linguagem será regular (note-se que o "Pumping Lemma" para linguagens regulares — Teorema 3.15 — é um caso particular do "Pumping Lemma" para linguagens livres de contexto). Se v e x forem simultaneamente diferentes da cadeia vazia, então a linguagem é livre de contexto e não-regular, e os termos v e x ocorrerão sempre de forma balanceada nas sentenças da linguagem.

Como no caso do "Pumping Lemma" para linguagens regulares, que estabelece uma propriedade inerente a toda e qualquer linguagem regular, o "Pumping Lemma" para

linguagens livres de contexto estabelece uma propriedade inerente a toda e qualquer linguagem livre de contexto. De maneira análoga àquele caso, uma de suas principais aplicações é na demonstração da existência de linguagens que não são livres de contexto, conforme mostrarão os exemplos a seguir.

Observe-se que o teorema garante a existência da constante n, mas a plicação do mesmo não exige que seja determinado o seu valor, como mostram os exemplos seguintes.

Exemplo 4.38 A linguagem $L_1 = \{a^k b^k c^k \mid k \geq 1\}$ não é livre de contexto. Suponha-se, por hipótese, que L_1 seja livre de contexto. De acordo com o Teorema 4.17 ("Pumping Lemma" para linguagens livres de contexto), existe uma constante inteira n tal que, qualquer que seja a sentença $\gamma \in L_1$, $\left|\gamma\right| \geq n$, então $\gamma = uvwxy$, $\left|vwx\right| \leq n$, $\left|vx\right| \geq 1$ e $uv^i wx^i y \in L_1$.

Seja $\gamma = a^n b^n c^n$. Como $\left|a^n b^n c^n\right| = 3 * n$, então está satisfeita a condição $\left|\gamma\right| \geq n$. Portanto, $\gamma = uvwxy$. Nesta situação, a subcadeia vwx pode assumir um dos seguintes formatos (lembrar que $\left|vwx\right| \leq n$):

1. vwx contém apenas símbolos "a" (pelo menos um); portanto, vx também contém apenas símbolos a (pelo menos um);

2. vwx contém apenas símbolos "b" (pelo menos um); portanto, vx também contém apenas símbolos b (pelo menos um);

3. vwx contém apenas símbolos "c" (pelo menos um); portanto, vx também contém apenas símbolos c (pelo menos um);

4. vwx contém símbolos "a" (pelo menos um) seguidos de símbolos "b" (pelo menos um); portanto, vx contém pelo menos um símbolo a ou b, porém nenhum símbolo c;

5. vwx contém símbolos "b" (pelo menos um) seguidos de símbolos "c" (pelo menos um); portanto, vx contém pelo menos um símbolo b ou c, porém nenhum símbolo a.

É impossível que a subcadeia vwx contenha simultaneamente símbolos "a", "b" e "c", uma vez que, para isso acontecer, seria necessário que o comprimento de vwx fosse no mínimo $n + 2$, o que contraria o "Pumping Lemma".

Passando-se à análise de cada uma dessas possibilidades, considerando o formato assumido pela sentença uwy, isto é, pela sentença $uvwxy$ após a remoção das subcadeias v e x, o que é previsto pelo "Lemma" quando estabelece que todas as sentenças $uv^i wx^i y$ devem pertencer a L_1 (neste caso, faz-se $i = 0$). Lembrar que $\left|vx\right| \geq 1$:

1. uwy conterá n símbolos "b", n símbolos "c" e uma quantidade de símbolos "a" menor que n; logo, uwy não pertence a L_1;

2. uwy conterá n símbolos "a", n símbolos "c" e uma quantidade de símbolos "b" menor que n; logo, uwy não pertence a L_1;

3. uwy conterá n símbolos "a", n símbolos "b" e uma quantidade de símbolos "c" menor que n; logo, uwy não pertence a L_1;

4. uwy conterá n símbolos "c", e quantidades de símbolos "a" e de símbolos "b" respectivamente menores que n; logo, uwy não pertence a L_1;

5. uwy conterá n símbolos "a", e quantidades de símbolos "b" e de símbolos "c" respectivamente menores que n; logo, uwy não pertence a L_1.

Portanto, a sentença $a^n b^n c^n$ contradiz a hipótese inicial, provando que a linguagem L_1 não é livre de contexto. Cumpre observar que, no Exemplo 3.50 da Seção 3.8 ("Pumping Lemma" para as Linguagens Regulares), esta mesma linguagem foi demonstrada como sendo não-regular. □

Exemplo 4.39 A linguagem $L_2 = \{wcw \mid w \in \{a, b\}^+\}$ não é livre de contexto. Esta linguagem sintetiza uma característica bastante comum nas linguagens de programação mais usuais: a necessidade

de se usarem identificadores idênticos em partes diferentes de um mesmo programa, por exemplo, na declaração de uma variável e na ocasião de sua utilização posterior.

Suponha-se, por hipótese, que L_2 seja livre de contexto. De acordo com o "Pumping Lemma" para linguagens livres de contexto, existe uma constante inteira n tal que, qualquer que seja a sentença $\gamma \in L_2$, $\left|\gamma\right| \geq n$, então $\gamma = uvwxy$, $\left|vwx\right| \leq n$, $\left|vx\right| \geq 1$ e $uv^iwx^iy \in L_2$.

Considere-se, por exemplo, a sentença $\gamma = a^nb^nca^nb^n$. Como $\left|\gamma\right| = 4 * n + 1 \geq n$, então está satisfeita a condição do "Pumping Lemma" para a escolha da sentença γ. Portanto, $\gamma = uvwxy$ e a subcadeia vwx pode assumir um dos seguintes formatos:

1. vwx contém apenas símbolos "a" (pelo menos um);
2. vwx contém apenas símbolos "b" (pelo menos um);
3. vwx contém símbolos "a" (pelo menos um) seguidos de símbolos "b" (pelo menos um);
4. vwx se inicia com símbolos "b" (zero ou mais) e termina com o símbolo "c";
5. vwx se inicia com o símbolo "c" e termina com símbolos "a" (zero ou mais);
6. vwx se inicia com símbolos "b" (pelo menos um) , continua com um símbolo "c" e termina com símbolos "a" (pelo menos um).

De maneira análoga ao que foi mostrado no Exemplo 4.38, passa-se a examinar o formato das cadeias uwy que são geradas em cada um desses casos:

1. Ao extrair símbolos "a" da subcadeia w que antecede o símbolo "c", esta resulta diferente da subcadeia w posterior ao mesmo "c"; o mesmo acontece se forem extraídos símbolos "a" da subcadeia posterior ao símbolo "c"; logo, uwy não pertence a L_2;
2. Semelhante ao caso anterior;
3. Modifica, exclusivamente, a subcadeia anterior ao símbolo "c" ou a subcadeia posterior ao símbolo "c"; logo, uwy não pertence a L_2;
4. Como $\left|vx\right| \geq 1$, vx contém pelo menos um símbolo "b" ou um símbolo "c", que, se removidos da sentença γ, provocam a geração de uma sentença que não pertence a L_2 (seja porque as cadeias w tornam-se diferentes, seja porque o símbolo "c" que as separa desaparece);
5. Semelhante ao caso anterior;
6. Como $\left|vx\right| \geq 1$, vx contém pelo menos um símbolo "b" (da primeira subcadeia w) ou um símbolo "c" ou, ainda, um símbolo "a" (da segunda subcadeia w); qualquer que seja o caso, a cadeia resultante uwy não pertence a L_2 (seja porque as cadeias w tornam-se diferentes, seja porque o símbolo "c" que as separa desaparece).

Pelo exposto, fica claro que a hipótese inicial é falsa e que L_2 não é uma linguagem livre de contexto. □

Exemplo 4.40 A linguagem $L_3 = \{a^k \mid k \geq 1$ é um número primo$\}$ não é livre de contexto. Suponha-se que L_3 seja livre de contexto e considere-se a sentença $\gamma = a^p$, cujo comprimento seja maior que o valor da constante n definida pelo "Pumping Lemma" para linguagens livres de contexto, portanto, $p > n$. Se $\gamma = uvwxy$ pertence a L_3, então, de acordo com o "Lemma", a sentença uwy também deve pertencer. Seja $q = \left|uwy\right|$.

O comprimento das sentenças uv^iwx^iy, portanto, pode ser calculado da seguinte forma: $\left|uv^iwx^iy\right| = \left|uwy\right| + i^* \left| vx\right|$. Em particular, o comprimento da sentença $\left|uv^qwx^qy\right| = \left|uwy\right| + q^* \left| vx\right| = q + q^* \left| vx\right| = q^*(1 + \left|vx\right|)$. Logo, o comprimento de $\left|uv^qwx^qy\right|$ não é primo e isso prova que L_3 não pode ser livre de contexto.

Vale lembrar que, no Exemplo 3.51 da Seção 3.8 ("Pumping Lemma" para as Linguagens Regulares), esta mesma linguagem foi demonstrada como sendo não-regular. □

4.11 Linguagens Livres de Contexto Determinísticas

Diferentemente da classe das linguagens regulares, que podem ser reconhecidas indistintamente por autômatos finitos, determinísticos ou não-determinísticos, somente autômatos de pilha não-determinísticos são capazes de reconhecer as linguagens livres de contexto, no caso geral.

Esta conclusão, que pode ser provada demonstrando-se a existência de linguagens livres de contexto não-reconhecíveis por autômatos de pilha determinísticos, quaisquer que sejam eles, possui uma importante conseqüência prática, uma vez que os autômatos de pilha não-determinísticos são por natureza pouco eficientes, e por isso constituem modelos de implementação pouco atraentes em diversas situações, entre as quais, na importante área de construção de compiladores.

Exemplo 4.41 A linguagem $L_1 = \{ww^R \mid w \in \{a, b\}^*\}$ é não-determinística, ou seja, é reconhecida apenas por autômatos de pilha não-determinísticos. Um exemplo de tal autômato, com critério de aceitação baseado em pilha vazia, é apresentado na Figura 4.22.

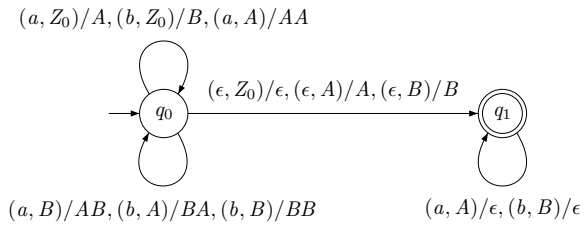

Figura 4.22: Autômato de pilha para a linguagem $\{ww^R \mid w \in \{a, b\}^*\}$

São exemplos de cadeias aceitas por este autômato aa e $abba$, como se pode verificar nos reconhecimentos abaixo:

- $(q_0, aa, Z_0) \vdash (q_0, a, A) \vdash (q_1, a, A) \vdash (q_1, \epsilon, \epsilon)$
- $(q_0, abba, Z_0) \vdash (q_0, bba, A) \vdash (q_0, ba, BA) \vdash (q_1, ba, BA) \vdash (q_1, a, A) \vdash (q_1, \epsilon, \epsilon)$

Como é fácil observar, a característica não-determinística do autômato é fundamental para que ele possa determinar o ponto exato da cadeia de entrada em que termina a subcadeia w e se inicia a subcadeia w^R: apenas uma movimentação em vazio no ponto central da cadeia será capaz de fazer com que o autômato atinja uma configração final. Tal característica é necessária, uma vez que o autômato, por conta de suas próprias limitações, não tem condições de determinar diretamente o ponto exato em que se inicia a cadeia w^R. □

Assim, apesar do elevado interesse prático exibido pelas linguagens livres de contexto, decorrente da relativa facilidade com que elas podem ser representadas e manipuladas, a dificuldade acima mencionada levou os pesquisadores a um estudo mais aprofundado dessa classe de linguagens, visando fornecer aos projetistas e implementadores de linguagens artificiais subsídios que permitissem o aproveitamento prático das características dessa classe de linguagens, sem no entanto sacrificar o desempenho dos respectivos reconhecedores pela presença de eventuais não-determinismos.

Como conseqüência, foi identificada e caracterizada a classe das **linguagens livres de contexto determinísticas**, que se provou ser subconjunto próprio das linguagens livres de contexto genéricas. Por definição, linguagens livres de contexto determinísticas são aquelas que podem ser reconhecidas por autômatos de pilha determinísticos.

Em outras palavras, uma linguagem livre de contexto é dita determinística se for possível demonstrar a existência de pelo menos um autômato de pilha determinístico que a reconheça.

A linguagem para a qual se pode provar a inexistência de quaisquer autômatos de pilha determinísticos que a reconheçam, e, simultaneamente, a existência de pelo menos um autômato de pilha não-determinístico que a reconheça, é denominada **linguagem livre de contexto não-determinística**, e o conjunto de tais linguagens constitui uma classe importante das linguagens livres de contexto gerais.

A classe das linguagens livres de contexto determinísticas, também conhecida como $LR(k)$, apresenta um elevado interesse prático, uma vez que propicia a obtenção de implementações eficientes para todas as linguagens livres de contexto determinísticas. Por outro lado, a restrição que as caracteriza felizmente não cria limitações sérias ao poder de representação de tal classe de linguagens, quando comparado ao das linguagens do tipo 2 genéricas, e como conseqüência as linguagens $LR(k)$ constituem uma classe de linguagens ainda muito aderente às necessidades práticas de representação sintática formal das linguagens de programação usuais, sendo por esse motivo largamente utilizadas na área de processamento de linguagens artificiais, em especial das linguagens de programação.

Por sua vez, também como parte da classe das linguagens $LR(k)$, é possível identificar uma nova classe de linguagens, constituída na forma de um subconjunto próprio, e que também apresenta elevado interesse prático: trata-se das linguagens $LL(k)$, e a sua caracterização decorre da possibilidade de obtenção de modelos de implementação ainda mais eficientes do que os obtidos para as linguagens $LR(k)$. No entanto, a sua abrangência é significativamente menor do que a das linguagens $LR(k)$, e por esse motivo o seu interesse prático não é tão grande como o das primeiras, sendo no entanto bastante útil no caso de linguagens livres de contexto determinísticas com menor grau de sofisticação sintática.

A seguir serão apresentadas as definições de gramáticas e de linguagens $LL(k)$ e $LR(k)$. Para maiores detalhes, exemplos, propriedades, algoritmos de verificação gramatical e técnicas de construção de reconhecedores para essas classes de linguagens, recomenda-se a leitura de [48] ou [49].

Suponha-se que $G = (V, \Sigma, P, S)$ seja uma gramática livre de contexto. Então:

- Diz-se que G é uma gramática $LL(k)$ se for possível analisar deterministicamente as cadeias de $L(G)$ da seguinte forma: quaisquer que sejam a cadeia de entrada $w \in L(G)$ e a forma sentencial $\alpha X \beta$, obtida por meio da aplicação exclusiva de derivações mais à esquerda, e tal que $S \Rightarrow^* \alpha X \beta$, com $\alpha \in \Sigma^*$, $X \in N$ e $\beta \in V^*$, se a escolha da produção $X \rightarrow \mu$ puder ser feita de forma unívoca, inspecionando-se no máximo os k primeiros símbolos de γ, onde $\alpha X \beta \Rightarrow \alpha \mu \beta \Rightarrow^* \alpha\gamma$, $\gamma \in \Sigma^*$. A aplicação da produção escolhida faz com que $\alpha X \beta \Rightarrow \alpha \mu \beta$ e, eventualmente, $\alpha \mu \beta \Rightarrow^* w$.

- Diz-se que G é uma gramática $LR(k)$ se for possível analisar deterministicamente as cadeias de $L(G)$ da seguinte forma: quaisquer que sejam a cadeia de entrada $w \in L(G)$ e a forma sentencial $\alpha \mu \beta$, obtida por meio da aplicação exclusiva de reduções[3] mais à esquerda, e tal que $w \Rightarrow^* \alpha\gamma \Rightarrow^* \alpha \mu \beta$, com $\alpha, \mu \in V^*$ e $\beta, \gamma \in \Sigma^*$, se a escolha da produção $X \rightarrow \mu$ puder ser feita de forma unívoca, inspecionando-

[3]Uma redução corresponde ao inverso de uma derivação, ou seja, à substituição do lado direito de uma produção, em uma forma sentencial, pelo lado esquerdo correspondente. Por exemplo, se $X \rightarrow \mu$ é uma produção, então $\alpha X \beta \Rightarrow \alpha \mu \beta$ denota uma derivação e $\alpha \mu \beta \Rightarrow \alpha X \beta$ denota uma redução. Uma seqüência de reduções mais à esquerda corresponde à ordem inversa da seqüência de derivações mais à direita que gera a mesma cadeia.

se no máximo os k primeiros símbolos de γ. A aplicação da produção escolhida faz com que $\alpha\,\mu\,\beta \Rightarrow \alpha X \beta$ e, eventualmente, $\alpha X \beta \Rightarrow^* S$.

Diz-se que uma **linguagem** é $LL(k)$ se ela for gerada por uma gramática $LL(k)$. Por outro lado, de acordo com a definição, toda gramática $LL(k)$ é, naturalmente, também uma gramática $LL(k+1), LL(k+2)$ etc. Logo, toda linguagem $LL(k)$ é também uma linguagem $LL(k+1), LL(k+2)$ etc. Na prática, no entanto, costuma-se atribuir a uma linguagem o menor valor de k entre todas as gramáticas que a geram. Considerações análogas se aplicam ao caso das gramáticas e das linguagens $LR(k)$.

Do ponto de vista da hierarquia de inclusão própria das linguagens formais, a classe das linguagens $LR(k)$ e a classe das linguagens $LL(k)$ situam-se entre a classe das linguagens regulares e a classe das linguagens livres de contexto genéricas, conforme ilustrado na Figura 4.23:

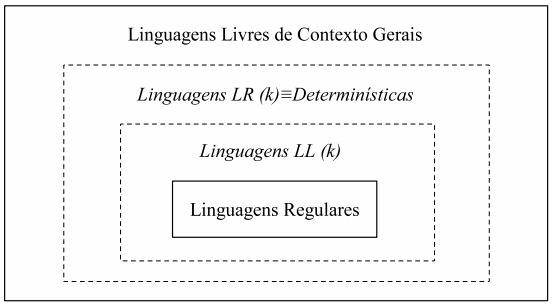

Figura 4.23: As subclasses $LL(k)$ e $LR(k)$

A demonstração da equivalência entre a classe das linguagens livres de contexto que podem ser reconhecidas deterministicamente e a classe das linguagens $LR(k)$ pode ser encontrada em [47]. A demonstração da existência de um algoritmo para determinar se uma dada gramática é $LR(k)$ para um certo valor de k também pode ser encontrada em [47].

Exemplo 4.42 A linguagem $L_2 = \{a^n b^n c \mid n \geq 0\}$ é $LL(1)$, porém não é regular. A não-regularidade pode ser demonstrada facilmente pela aplicação do "Pumping Lemma" para as linguagens regulares. A gramática livre de contexto G_2 abaixo gera esta linguagem:

$$\begin{aligned} G_2 &= (V, \Sigma, P, S) \\ V &= \{S, X, a, b, c\} \\ \Sigma &= \{a, b, c\} \\ P &= \{S \to Xc, X \to aXb \mid \epsilon\} \end{aligned}$$

Quaisquer que sejam a forma sentencial e a cadeia de entrada consideradas, o seguinte critério permite determinar univocamente a produção a ser aplicada:

- Se a forma sentencial corrente for S:
  - Se o terminal corrente for a, escolhe-se a produção $S \to Xc$;
  - Se o terminal corrente for b, há um erro na cadeia de entrada;
  - Se o terminal corrente for c, escolhe-se a produção $S \to Xc$;

- Se a forma sentencial corrente for Xc:
  - Se o terminal corrente for a, escolhe-se a produção $X \rightarrow aXb$;
  - Se o terminal corrente for b, há um erro na cadeia de entrada;
  - Se o terminal corrente for c, escolhe-se a produção $X \rightarrow \epsilon$;
- Se a forma sentencial corrente for $a^i X b^i c, \forall\, i \geq 1$:
  - Se o terminal corrente for a, escolhe-se a produção $X \rightarrow aXb$;
  - Se o terminal corrente for b, escolhe-se a produção $X \rightarrow \epsilon$;
  - Se o terminal corrente for c, há um erro na cadeia de entrada.

Logo, G_2 é $LL(1)$ — $LL(k)$ para $k = 1$ — pois, dada uma forma sentencial, basta inspecionar apenas um símbolo da cadeia de entrada para determinar univocamente a regra a ser aplicada. Conseqüentemente, $L(G_2)$ também é $LL(1)$.

Considerem-se as cadeias $aabbc$ e c. As Tabelas 4.1 e 4.2 ilustram, respectivamente, as seqüências de derivações mais à esquerda que refletem a operação de um reconhecedor $LL(1)$ na aceitação das mesmas. Os delimitadores "[" e "]" indicam o símbolo corrente da cadeia de entrada.

Forma sentencial	Cadeia de entrada	Produção utilizada
S	$[a]abbc$	$S \rightarrow Xc$
Xc	$[a]abbc$	$X \rightarrow aXb$
$aXbc$	$[a]abbc$	-
$aXbc$	$a[a]bbc$	$X \rightarrow aXb$
$aaXbbc$	$a[a]bbc$	-
$aaXbbc$	$aa[b]bc$	$X \rightarrow \epsilon$
$aabbc$	$aa[b]bc$	-
$aabbc$	$aab[b]c$	-
$aabbc$	$aabb[c]$	-
$aabbc$	$aabbc$	-

Tabela 4.1: Seqüência de derivações mais à esquerda realizada por um reconhecedor $LL(1)$ na análise da cadeia $aabbc$

Forma sentencial	Cadeia de entrada	Produção utilizada
S	$[c]$	$S \rightarrow Xc$
Xc	$[c]$	$X \rightarrow \epsilon$
c	$[c]$	-
c	c	-

Tabela 4.2: Seqüência de derivações mais à esquerda realizada por um reconhecedor $LL(1)$ na análise da cadeia c

□

Exemplo 4.43 Considere-se a linguagem $L_3 = \{x^{2n} y^{2n} e \mid n \geq 0\} \cup \{x^{2n+1} y^{2n+1} o \mid n \geq 0\}$. A gramática G_3 abaixo gera esta linguagem:

$$\begin{aligned} G_3 &= (V, \Sigma, P, S) \\ V &= \{S, E, O, x, y, e, o\} \\ \Sigma &= \{x, y, e, o\} \\ P &= \{S \rightarrow Ee \mid Oo, E \rightarrow xOy \mid \epsilon, O \rightarrow xEy\} \end{aligned}$$

Justifica-se que a linguagem não é $LL(k)$ porque, para n genérico, não há um valor de k que seja suficiente para garantir a escolha da produção correta a ser aplicada na gramática, qualquer que seja ela. Fixe-se um valor arbitrário κ para k, ou seja, $k = \kappa$. Basta escolher uma cadeia de entrada em que $2n \geq \kappa$ para que não se possa determinar univocamente a produção a ser aplicada. Em outras palavras, qualquer que seja o valor de k selecionado, sempre haverá uma cadeia pertencente à linguagem tal que a quantidade de símbolos x é maior ou igual a k, inviabilizando o reconhecimento da mesma como sendo pertencente ao conjunto $\{x^{2n}y^{2n}e\}$ ou ao conjunto $\{x^{2n+1}y^{2n+1}o\}$.

Quaisquer que sejam a forma sentencial e a cadeia de entrada consideradas, o seguinte critério permite determinar univocamente a produção a ser aplicada:

- Se a forma sentencial corrente for $x^{2n}y^{2n}e, n \geq 0$ ou $x^{2n+1}y^{2n+1}o, n \geq 0$:
  - Se o terminal corrente for y, escolhe-se a produção $E \rightarrow \epsilon$;
  - Se o terminal corrente for e, escolhe-se a produção $E \rightarrow \epsilon$;
  - Se o terminal corrente for o, há um erro na cadeia de entrada;
- Se a forma sentencial corrente for $x^{2n}Ey^{2n}e, n \geq 0$ ou $x^{2n+1}Ey^{2n+1}o, n \geq 0$:
  - Escolhe-se a produção $O \rightarrow xEy$;
- Se a forma sentencial corrente for $x^{2n}Oy^{2n}o, n \geq 0$ ou $x^{2n+1}Oy^{2n+1}e, n \geq 0$:
  - Escolhe-se a produção $E \rightarrow xOy$;
- Se a forma sentencial corrente for Ee:
  - Escolhe-se a produção $S \rightarrow Ee$;
- Se a forma sentencial corrente for Oo:
  - Escolhe-se a produção $S \rightarrow Oo$.

Logo, G_3 é $LR(1)$, ou seja, $LR(k)$ para $k = 1$, pois, dada uma forma sentencial, basta inspecionar apenas um símbolo da cadeia de entrada para determinar univocamente a regra a ser aplicada. Conseqüentemente, $L(G_3)$ também é $LR(1)$.

As Tabelas 4.3 e 4.3 ilustram as seqüências de reduções mais à esquerda que refletem a operação de um reconhecedor $LR(1)$ na aceitação das cadeias $xxyye$ e $xxxyyyo$, respectivamente. Os delimitadores "[" e "]" indicam o símbolo corrente da cadeia de entrada.

Forma sentencial	Cadeia de entrada	Produção utilizada
$xxyye$	$[x]xyye$	-
$xxyye$	$x[x]yye$	-
$xxyye$	$xx[y]ye$	$E \rightarrow \epsilon$
$xxEyye$	$xx[y]ye$	$O \rightarrow xEy$
$xOye$	$xxy[y]e$	$E \rightarrow xOy$
Ee	$xxyy[e]$	$S \rightarrow Ee$
S	$xxyye$	-

Tabela 4.3: Seqüência de reduções mais à esquerda realizada por um reconhecedor $LR(1)$ na análise da cadeia $xxyye$

Forma sentencial	Cadeia de entrada	Produção utilizada
$xxxyyyo$	$[x]xxyyyo$	-
$xxxyyyo$	$x[x]xyyyo$	-
$xxxyyyo$	$xx[x]yyyo$	-
$xxxyyyo$	$xxx[y]yyo$	$E \rightarrow \epsilon$
$xxxEyyyo$	$xxx[y]yyo$	$O \rightarrow xEy$
$xxOyyo$	$xxxy[y]yo$	$E \rightarrow xOy$
$xEyo$	$xxxyy[y]o$	$O \rightarrow xEy$
Oo	$xxxyyy[o]$	$S \rightarrow Oo$
S	$xxxyyyo$	-

Tabela 4.4: Seqüência de reduções mais à esquerda realizada por um reconhecedor $LR(1)$ na análise da cadeia $xxxyyyo$

□

Os Algoritmos de reconhecimento $LL(k)$ e $LR(k)$ apresentados nos Exemplos 4.42 e 4.43 foram elaborados com base em técnicas cuja apresentação e discussão fogem ao escopo deste texto. O assunto, no entanto, é exaustivamente estudado na literatura sobre compiladores e processadores de linguagens. Para mais detalhes, recomenda-se a leitura de [28], [49], [48], [24], [26], [27] ou [29].

Gramáticas e linguagens livres de contexto são extensivamente empregadas na definição e na implementação de linguagens artificiais, em particular das de programação. Aninhamentos, ou construções sintáticas que se repetem integralmente como partes de si próprias, são muito comuns em linguagens de programação convencionais, manifestando-se geralmente na especificação do fluxo de controle (comandos e blocos aninhados), das abstrações (procedimentos e funções aninhadas), das expressões (através do uso de parênteses aninhados) etc. Tal característica torna o uso das linguagens (e das gramáticas) livres de contexto determinísticas muito atraente, uma vez que elas permitem a obtenção de representações formais e de modelos de implementação que incorporam diretamente, e a um custo relativamente baixo, a especificação e a análise de construções sintáticas aninhadas.

Por esse motivo, as linguagens e as gramáticas livres de contexto determinísticas, sejam elas $LR(k)$ ou $LL(k)$, costumam ser empregadas na formalização sintática da maioria das linguagens de programação convencionais. Como conseqüência, os compiladores

dessas linguagens costumam ser construídos com base nos correspondentes autômatos de pilha determinísticos. Tais reconhecedores, que servem como ponto de partida para a construção de analisadores sintáticos, são com freqüência usados como núcleos dos compiladores das respectivas linguagens, e a eles são agregados os demais módulos de análise e de geração de código, para completar as funções do compilador.

4.12 Linguagens Livres de Contexto Não-Ambíguas

O estudo sistemático das gramáticas e das linguagens livres de contexto, realizado nas décadas de 1960 e 1970, teve como principal finalidade a obtenção de reconhecedores eficientes para classes de linguagens suficientemente abrangentes, de forma a permitir a sua aplicação prática na construção de compiladores para linguagens de programação de alto nível comerciais.

Entre os principais resultados obtidos naquela época, encontra-se uma classificação hierárquica das gramáticas livres de contexto, criada a partir de especializações do modelo geral de gramática livre de contexto apresentado na Seção 4.1. Tais especializações, apresentadas e discutidas em [49] e [25], definem, entre outras, as classes das gramáticas $LL(k)$ e das gramáticas $LR(k)$, discutidas na Seção 4.11, e também a importante classe das gramáticas não-ambíguas, conforme ilustrado na Figura 4.24.

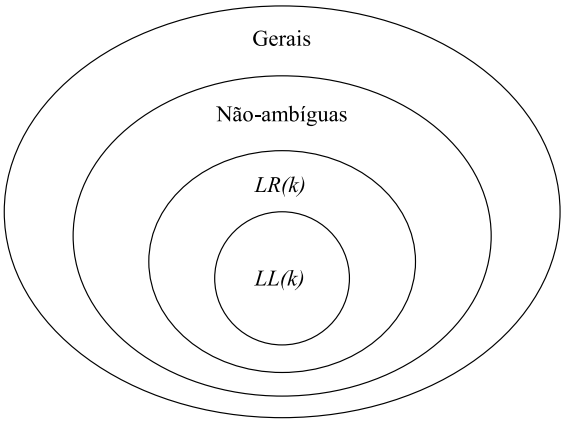

Figura 4.24: Hierarquia das gramáticas livres de contexto

Conforme mencionado na Seção 4.11, a classe das gramáticas $LR(k)$ corresponde exatamente à classe das linguagens determinísticas. Por outro lado, conforme definido na Seção 4.1, a classe das gramáticas livres de contexto gerais, sobre as quais não se aplica qualquer restrição adicional, corresponde à classe das linguagens livres de contexto gerais. A Figura 4.25 ilustra tais relações.

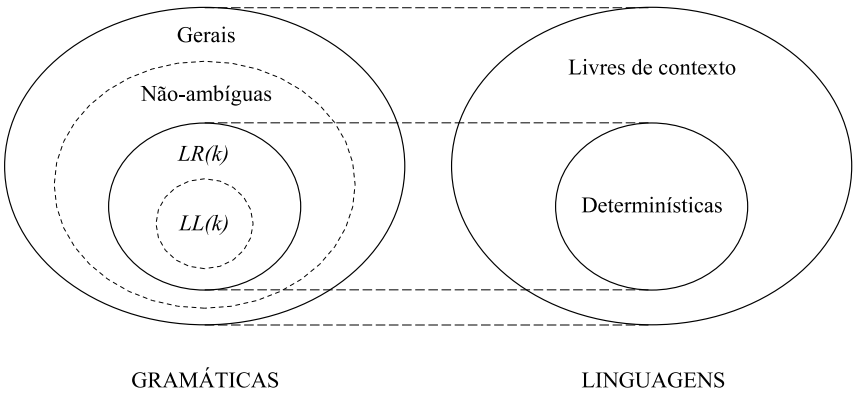

Figura 4.25: Correspondência entre classes de gramáticas e classes de linguagens livres de contexto

Conforme discutido em [49], demonstra-se que a relação de inclusão entre as classes de gramáticas representadas na Figura 4.24 é própria. Isso significa que é possível garantir a existência, já verificada na Seção 4.4, de gramáticas livres de contexto que não pertencem à classe das gramáticas livres de contexto não-ambíguas — sendo, portanto, ambíguas —, e também de gramáticas livres de contexto que não são $LR(k)$, porém ainda assim são não-ambíguas.

Dessa forma, cumpre responder à questão: *"Será possível identificar alguma subclasse das linguagens livres de contexto que esteja em correspondência com a classe das gramáticas livres de contexto não-ambíguas?"*.

Para responder a essa questão, considere-se:

1. Conforme a definição, toda e qualquer linguagem livre de contexto que não seja inerentemente ambígua (ver Seção 4.4) pode ser gerada por pelo menos uma gramática livre de contexto não-ambígua. Portanto, é razoável considerar-se uma classe de linguagens livres de contexto não-ambíguas.

2. De acordo com a hierarquia de gramáticas apresentada na Figura 4.24, toda gramática livre de contexto $LR(k)$ é também uma gramática livre de contexto não-ambígua, o que implica que a classe das linguagens livres de contexto determinísticas está incluída na classe das linguagens livres de contexto não-ambíguas.

3. Considere-se a linguagem $\{ww^R \mid w \in \{a, b\}^* \}$, que pode ser gerada pela gramática não-ambígua $G = (\{S, a, b, \}, \{a, b\}, \{S \rightarrow aSa \mid bSb \mid \epsilon\}, S)$, mas não pode ser reconhecida por nenhum autômato de pilha determinístico (ver Exemplo 4.41). Trata-se, portanto, de uma linguagem livre de contexto não-ambígua, porém não-determinística, e a sua existência demonstra que a relação de inclusão entre a classe das linguagens determinísticas e a classe das linguagens não-ambíguas é própria.

4. Ainda de acordo com a hierarquia de gramáticas apresentada na Figura 4.24, toda gramática livre de contexto não-ambígua é também uma gramática livre de contexto geral, o que implica que a classe das linguagens livres de contexto não-ambíguas está incluída na classe das linguagens livres de contexto gerais.

5. As linguagens inerentemente ambíguas, de acordo com a definição apresentada na Seção 4.4, são aquelas para as quais inexistem gramáticas livres de contexto não-ambíguas que as definam. Tais linguagens são definidas exclusivamente através de gramáticas livres de contexto ambíguas. Portanto, existem linguagens livres de contexto gerais que não podem ser consideradas não-ambíguas, e a relação de inclusão entre essas duas classes de linguagens também é própria. Um exemplo é a linguagem inerentemente ambígua $\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$, do Exemplo 4.15.

Logo, é natural concluir que a resposta para a pergunta anterior é: *"Sim, e ela corresponde à classe das linguagens livres de contexto não inerentemente ambíguas ou, simplesmente, não-ambíguas"*.

Em outras palavras, a classe das **linguagens livres de contexto não-ambíguas** está incluída propriamente na classe das linguagens livres de contexto gerais, e a classe das linguagens livres de contexto que não são não-ambíguas corresponde à classe das linguagens livres de contexto que são inerentemente ambíguas. A Figura 4.26 ilustra essas relações.

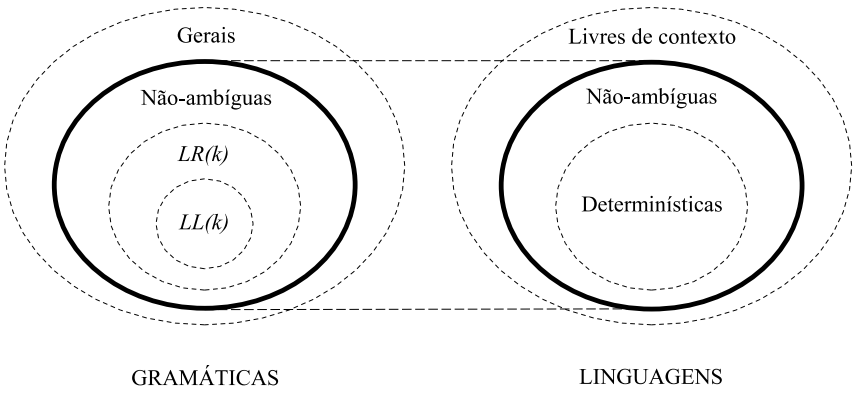

Figura 4.26: A classe das linguagens livres de contexto não-ambíguas

Tal resultado permite chegar às seguintes conclusões adicionais:

- Toda linguagem livre de contexto determinística é também não-ambígua, ou seja, se existe um autômato de pilha determinístico que reconhece uma linguagem, então existe também uma gramática livre de contexto não-ambígua que a gera. De forma equivalente, pode-se também dizer que toda gramática $LR(k)$ é também não-ambígua.

- Nem toda linguagem livre de contexto não-ambígua é determinística. Em outras palavras, existem linguagens livres de contexto que podem ser geradas por gramáticas não-ambíguas, mas que podem ser reconhecidas apenas através de autômatos de pilha não-determinísticos. De outra forma, pode-se também afirmar que certas linguagens livres de contexto podem apenas ser geradas por gramáticas não-ambíguas que não são $LR(k)$.

- Toda linguagem inerentemente ambígua é também não-determinística. Isso significa que a inexistência de uma gramática não-ambígua que gera determinada linguagem implica a inexistência de um autômato de pilha determinístico que reconheça essa mesma linguagem. Implica, também, a inexistênca de uma gramática $LR(k)$ que a gere.

A relação entre a classe das linguagens livres de contexto e as diversas subclasses nela contidas, conforme discutido na Seção 4.11 e também na presente seção, está representada de forma resumida na Figura 4.27.

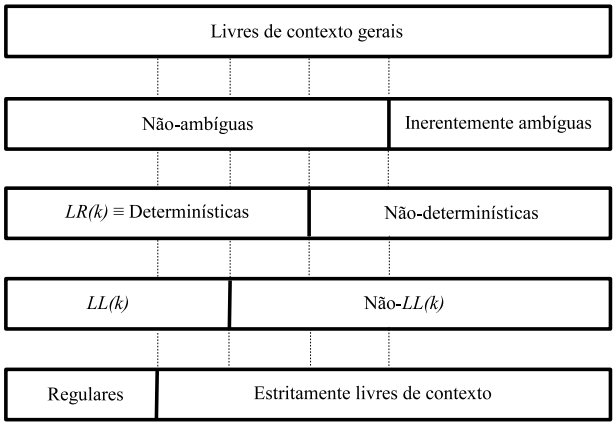

Figura 4.27: Relação entre as diversas subclasses das linguagens livres de contexto

A existência de um subconjunto das linguagens livres de contexto que seja simultaneamente não-ambíguo e determinístico, e que ainda assim seja suficientemente abrangente para representar uma vasta gama de linguagens de programação de alto nível comerciais, é um dos resultados mais marcantes da teoria das linguagens livres de contexto. Ele serviu de base para a rápida consolidação e avanço tecnológico verificados, nas décadas de 1960 e 1970, nas áreas de especificação de linguagens de programação e de construção de seus processadores (compiladores e interpretadores).

4.13 Propriedades de Fechamento

As linguagens livres de contexto são fechadas em relação às operações de união, concatenação e fecho de Kleene. Elas não são fechadas em relação às operações de complemento e intersecção. Os teoremas seguintes apresentam as respectivas demonstrações.

Teorema 4.18 (Fecho na união) *As linguagens livres de contexto são fechadas em relação à operação de união.*

Justificativa Sejam L_1 e L_2 duas linguagens livres de contexto quaisquer. Então, $L_1 = L_1(G_1)$ e $L_2 = L_2(G_2)$, G_1 e G_2 sendo duas gramáticas livres de contexto: $G_1 = (V_1, \Sigma_1, P_1, S_1)$ e $G_2 = (V_2, \Sigma_2, P_2, S_2)$. Admita-se que $N_1 \cap N_2 = \varnothing$. Se isso não for verdade, os símbolos não-terminais podem ser facilmente renomeados, sem prejuízo para as linguagens que estiverem sendo definidas.

A linguagem $L_3 = L_1 \cup L_2$ é gerada pela gramática G_3, ou seja, $L_3 = L_3(G_3)$, com $G_3 = (V_3, \Sigma_3, P_3, S_3)$, onde:

- $V_3 = V_1 \cup V_2 \cup \{S_3\}$
- $\Sigma_3 = \Sigma_1 \cup \Sigma_2$
- $P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_1, S_3 \rightarrow S_2\}$

Como, por hipótese e por construção, as regras contidas em P_3 atendem, todas, às condições necessárias para uma gramática ser classificada como livre de contexto, segue que G_3 é livre de contexto e, conseqüentemente, L_3 é uma linguagem livre de contexto■

Teorema 4.19 (Fecho na concatenação) *As linguagens livres de contexto são fechadas em relação à operação de concatenação.*

Justificativa Sejam L_1 e L_2 duas linguagens livres de contexto quaisquer. Então, $L_1 = L_1(G_1)$ e $L_2 = L_2(G_2)$, G_1 e G_2 sendo duas gramáticas livres de contexto: $G_1 = (V_1, \Sigma_1, P_1, S_1)$ e $G_2 = (V_2, \Sigma_2, P_2, S_2)$. Admita-se que $N_1 \cap N_2 = \varnothing$. Se isso não for verdade, os símbolos não-terminais podem ser facilmente renomeados, sem prejuízo para as linguagens que estiverem sendo definidas.

A linguagem $L_3 = L_1 \cup L_2$ é gerada pela gramática G_3, ou seja, $L_3 = L_3(G_3)$, com $G_3 = (V_3, \Sigma_3, P_3, S_3)$, onde:

- $V_3 = V_1 \cup V_2 \cup \{S_3\}$
- $\Sigma_3 = \Sigma_1 \cup \Sigma_2$
- $P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_1 S_2\}$

Como, por hipótese e por construção, as regras contidas em P_3 atendem, todas, às condições necessárias para uma gramática ser classificada como livre de contexto, segue que G_3 é livre de contexto e, conseqüentemente, L_3 é uma linguagem livre de contexto■

Teorema 4.20 (Fecho no fecho) *As linguagens livres de contexto são fechadas em relação à operação de fecho de Kleene.*

Justificativa Seja L_1 uma linguagem livre de contexto qualquer. Então, $L_1 = L_1(G_1)$, G_1 sendo uma gramática livre de contexto: $G_1 = (V_1, \Sigma_1, P_1, S_1)$.

A linguagem $L_2 = L_1^*$ é gerada pela gramática G_2, ou seja, $L_2 = L_2(G_2)$, com $G_2 = (V_2, \Sigma_2, P_2, S_2)$, onde:

- $V_2 = V_1 \cup \{S_2\}$
- $\Sigma_2 = \Sigma_1$
- $P_2 = P_1 \cup \{S_2 \rightarrow S_1 S_2, S_2 \rightarrow \epsilon\}$

Como, por hipótese e por construção, as regras contidas em P_2 atendem, todas, às condições necessárias para uma gramática ser classificada como livre de contexto, segue que G_2 é livre de contexto e, conseqüentemente, L_2 é uma linguagem livre de contexto■

Teorema 4.21 (Fecho na intersecção) *As linguagens livres de contexto não são fechadas em relação à operação de intersecção.*

Justificativa É suficiente, para provar este teorema, que existem pelo menos duas linguagens livres de contexto cuja intersecção gera uma linguagem que não seja livre de contexto.

Considerem-se as linguagens $L_1 = \{a^m b^m c^n \mid m \geq 1, n \geq 1\}$ e $L_2 = \{a^n b^m c^m \mid m \geq 1, n \geq 1\}$. É fácil demonstrar que essas linguagens são geradas, respectivamente, pelas gramáticas livres de contexto G_1 e G_2:

$$\begin{aligned} G_1 &= (\{a, b, c, S, X, Y\}, \{a, b, c\}, \\ &\quad \{S \to XY, X \to aXb, X \to ab, Y \to cY, Y \to c\}, S) \\ G_2 &= (\{a, b, c, S, X, Y\}, \{a, b, c\}, \\ &\quad \{S \to XY, X \to aX, X \to a, Y \to bYc, Y \to bc\}, S) \end{aligned}$$

Por outro lado, a linguagem $L_3 = L_1 \cap L_2 = \{a^m b^m c^m \mid m \geq 1\}$ é uma linguagem que não é livre de contexto, conforme ilustrado na aplicação do "Pumping Lemma" para linguagens livres de contexto (ver Exemplo 4.38). ■

Não obstante este resultado, é possível demonstrar que a intersecção de uma linguagem livre de contexto com uma linguagem regular gera sempre uma linguagem livre de contexto.

Teorema 4.22 (Intersecção com linguagem regular) *Sejam L_1 e L_2, respectivamente, uma linguagem regular e uma linguagem livre de contexto. Então, $L_1 \cap L_2$ é uma linguagem livre de contexto.*

Justificativa Sejam M_1 um autômato finito determinístico e M_2 um autômato de pilha que reconhecem, respectivamente, as linguagens L_1 e L_2. A linguagem $L_3 = L_1 \cap L_2$ é reconhecida pelo autômato de pilha M_3 construído conforme o Algoritmo 4.14.

Algoritmo 4.14 (Intersecção com linguagem regular) *Obtenção de um autômato de pilha que reconhece a linguagem correspondente à intersecção de uma linguagem livre de contexto com uma linguagem regular.*

- Entradas:
  - um autômato finito determinístico $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$;
  - um autômato de pilha $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, q_{02}, Z_0, F_2)$;
- Saída: um autômato de pilha $M_3 = (Q_3, \Sigma, \Gamma, \delta_3, q_{03}, Z_0, F_3)$, com critério de aceitação de pilha vazia, tal que $V(M_3) = L(M_1) \cap L(M_2)$;
- Método:
  1. $Q_3 = Q_1 \times Q_2$;
  2. $q_{03} = (q_{01}, q_{02})$;

3. $F_3 = F_1 \times F_2$;

4. δ_3 definida como:

   a) $\delta_3((q_i, q_m), \sigma, X) = \{((q_j, q_n), \gamma)\}$,
   para $\delta_1(q_i, \sigma) = q_j$ e $\delta_2(q_m, \sigma, X) = \{(q_n, \gamma)\}$;

   b) $\delta_3((q_i, q_m), \epsilon, X) = \{((q_i, q_n), \gamma)\}$,
   para $\delta_2(q_m, \epsilon, X) = \{(q_n, \gamma)\}$.

É possível demonstrar (ver [46]), por indução sobre o tamanho das cadeias analisadas pelos autômatos, que $((q_{01}, q_{02}), w, Z_0) \vdash^* ((q_r, q_s), \epsilon, \epsilon), (q_r, q_s) \in F_3$, se e somente se $(q_{01}, w) \vdash^* (q_s, \epsilon), q_s \in F_1$ e $(q_{02}, w, Z_0) \vdash^* (q_r, \epsilon, \epsilon),\ q_r \in F_2$.

Em outras palavras, M_3 aceita w se e apenas se M_1 e M_2 também aceitam w. Se M_1 e/ou M_2 rejeitam w, então M_3 também rejeita w. Logo, L_3 corresponde à intersecção das linguagens L_1 e L_2, e a existência de M_3 prova que L_3 é uma linguagem livre de contexto.

Este teorema mostra como construir um autômato de pilha M_3 que simula, de maneira simultânea, as configurações que podem ser assumidas por um outro autômato de pilha M_2 e um autômato finito determinístico M_1 no reconhecimento de uma mesma cadeia de entrada.

Assim, os estados de M_3 correspondem ao produto cartesiano $Q_1 \times Q_2$. Se M_3 se encontrar no estado (q_i, q_m), isso significa que, naquela situação, M_1 encontrar-se-á no estado q_i e M_2 no estado q_m, se estivessem operando separadamente no reconhecimento da mesma cadeia de entrada.

Se a entrada σ for aceita simultaneamente no estado q_i de M_1 e no estado q_m de M_2, então uma nova transição é acrescentada em M_3 com o símbolo σ partindo do estado (q_i, q_m) e com destino no estado $(\delta_1(q_i), \delta_2(q_m))$.

Transições sem consumo de símbolo na cadeia de entrada de M_2 — da forma $\delta_2(q_i, \epsilon, X) = \{(q_j, \gamma)\}$ — podem ser simuladas sem dificuldade em M_3, bastando para isso "preservar" o estado corrente de M_1 em M_3, uma vez que M_1 é, por definição, determinístico, isento de transições em vazio.

Pelo fato de M_1 ser um autômato finito, que não necessita de pilha para operar, esta é manipulada por M_3 de forma a reproduzir a manipulação executada de forma isolada por M_2.

A cadeia de entrada será aceita se M_3 atingir um estado final (composto por estados finais de M_1 e de M_2) com a cadeia de entrada esgotada e com a pilha vazia. Nesta configuração ocorre a aceitação simultânea da cadeia de entrada por M_1 e M_2. ■

Exemplo 4.44 Sejam $L_1 = (ab^* \mid ba^*)$ e $L_2 = (a^n b^{n+1} \mid b^n a^{n+1})$, respectivamente reconhecidas pelo autômato finito M_1 e pelo autômato de pilha M_2 mostrados nas Figuras 4.28 e 4.29.

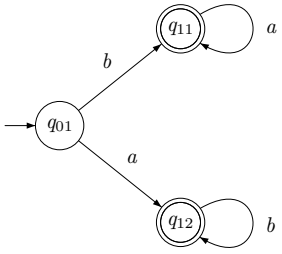

Figura 4.28: Autômato finito que reconhece $L_1 = (ab^* \mid ba^*)$

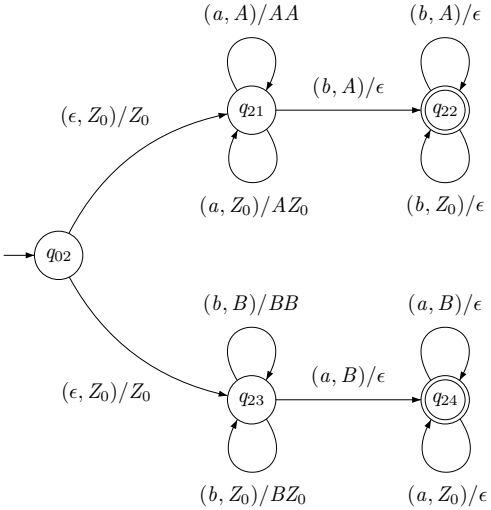

Figura 4.29: Autômato de pilha que reconhece $L_2 = (a^n b^{n+1} \mid b^n a^{n+1})$

A linguagem $L_3 = L_1 \cap L_2$ é reconhecida pelo autômato de pilha M_3, construído conforme o Algoritmo 4.14, utilizado na demonstração do Teorema 4.22:

$$
\begin{aligned}
Q_3 &= \{(q_{01}, q_{02}), (q_{01}, q_{21}), (q_{01}, q_{22}), (q_{01}, q_{23}), (q_{01}, q_{24}),\\
&\quad (q_{11}, q_{02}), (q_{11}, q_{21}), (q_{11}, q_{22}), (q_{11}, q_{23}), (q_{11}, q_{24}),\\
&\quad (q_{12}, q_{02}), (q_{12}, q_{21}), (q_{12}, q_{22}), (q_{12}, q_{23}), (q_{12}, q_{24})\}\\
F_3 &= \{(q_{11}, q_{22}), (q_{11}, q_{24}), (q_{12}, q_{22}), (q_{12}, q_{24})\}\\
\delta_3 &= \{((q_{01}, q_{02}), \epsilon, Z_0) \rightarrow ((q_{01}, q_{21}), Z_0),\\
&\quad ((q_{01}, q_{02}), \epsilon, Z_0) \rightarrow ((q_{01}, q_{23}), Z_0),\\
&\quad ((q_{01}, q_{21}), a, Z_0) \rightarrow ((q_{12}, q_{21}), AZ_0),\\
&\quad ((q_{01}, q_{21}), a, A) \rightarrow ((q_{12}, q_{21}), AA),\\
&\quad ((q_{01}, q_{21}), b, A) \rightarrow ((q_{11}, q_{22}), \epsilon),\\
&\quad ((q_{01}, q_{22}), b, Z_0) \rightarrow ((q_{11}, q_{22}), \epsilon),
\end{aligned}
$$

$$
\begin{aligned}
&((q_{01}, q_{22}), b, A) \rightarrow ((q_{11}, q_{22}), \epsilon),\\
&((q_{01}, q_{23}), a, B) \rightarrow ((q_{12}, q_{24}), \epsilon),\\
&((q_{01}, q_{23}), b, Z_0) \rightarrow ((q_{11}, q_{23}), BZ_0),\\
&((q_{01}, q_{23}), b, B) \rightarrow ((q_{11}, q_{23}), BB),\\
&((q_{01}, q_{24}), a, Z_0) \rightarrow ((q_{12}, q_{24}), \epsilon),\\
&((q_{01}, q_{24}), a, B) \rightarrow ((q_{12}, q_{24}), \epsilon),\\
&((q_{11}, q_{02}), \epsilon, Z_0) \rightarrow ((q_{11}, q_{21}), Z_0),\\
&((q_{11}, q_{02}), \epsilon, Z_0) \rightarrow ((q_{11}, q_{23}), Z_0),\\
&((q_{11}, q_{21}), a, Z_0) \rightarrow ((q_{11}, q_{21}), AZ_0),\\
&((q_{11}, q_{21}), a, A) \rightarrow ((q_{11}, q_{21}), AA),\\
&((q_{11}, q_{23}), a, B) \rightarrow ((q_{11}, q_{24}), \epsilon),\\
&((q_{11}, q_{24}), a, B) \rightarrow ((q_{11}, q_{24}), \epsilon),\\
&((q_{11}, q_{24}), a, Z_0) \rightarrow ((q_{11}, q_{24}), \epsilon),\\
&((q_{12}, q_{02}), \epsilon, Z_0) \rightarrow ((q_{12}, q_{21}), Z_0),\\
&((q_{12}, q_{02}), \epsilon, Z_0) \rightarrow ((q_{12}, q_{23}), Z_0),\\
&((q_{12}, q_{21}), b, A) \rightarrow ((q_{12}, q_{22}), \epsilon),\\
&((q_{12}, q_{22}), b, Z_0) \rightarrow ((q_{12}, q_{22}), \epsilon),\\
&((q_{12}, q_{22}), b, A) \rightarrow ((q_{12}, q_{22}), \epsilon),\\
&((q_{12}, q_{23}), b, Z_0) \rightarrow ((q_{12}, q_{24}), BZ_0),\\
&((q_{12}, q_{23}), b, B) \rightarrow ((q_{12}, q_{24}), BB)\}
\end{aligned}
$$

O autômato resultante M_3, excluídos os estados inúteis e inacessíveis, e com os demais estados renomeados, é representado na Figura 4.30. Note-se que a linguagem resultante, além de livre de contexto, é também regular. Por essa razão, o autômato de pilha resultante foi convertido para um autômato finito equivalente.

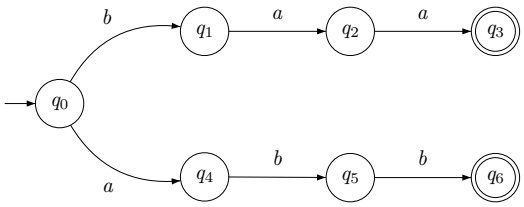

Figura 4.30: Autômato finito que reconhece a intersecção de L_1 com L_2

A linguagem reconhecida por M_3 é $(abb \mid baa)$, portanto livre de contexto, regular e finita. Observe-se e compare-se o reconhecimento destas duas sentenças em M_1, M_2 e M_3. Note-se que a configuração final assumida por M_3 compreende as configurações finais assumidas por M_1 e por M_2 para uma mesma cadeia de entrada.

- M_1:
  - $(q_{01}, abb) \vdash (q_{12}, bb) \vdash (q_{12}, b) \vdash (q_{12}, \epsilon)$
  - $(q_{01}, baa) \vdash (q_{11}, aa) \vdash (q_{11}, a) \vdash (q_{11}, \epsilon)$
- M_2:
  - $(q_{02}, abb, Z_0) \vdash (q_{21}, abb, Z_0) \vdash (q_{21}, bb, AZ_0) \vdash (q_{22}, b, Z_0) \vdash (q_{22}, \epsilon, \epsilon)$

  - $(q_{02}, baa, Z_0) \vdash (q_{23}, baa, Z_0) \vdash (q_{23}, aa, BZ_0) \vdash (q_{24}, a, Z_0) \vdash (q_{24}, \epsilon, \epsilon)$

- M_3:

  - $((q_{01}, q_{02}), abb, Z_0) \vdash ((q_{01}, q_{21}), abb, Z_0) \vdash ((q_{12}, q_{21}), bb, AZ_0) \vdash ((q_{12}, q_{22}), b, Z_0) \vdash ((q_{12}, q_{22}), \epsilon, \epsilon)$

  - $((q_{01}, q_{02}), baa, Z_0) \vdash ((q_{01}, q_{23}), baa, Z_0) \vdash ((q_{11}, q_{23}), aa, BZ_0) \vdash ((q_{11}, q_{24}), a, Z_0) \vdash ((q_{11}, q_{24}), \epsilon, \epsilon)$

□

Teorema 4.23 (Fecho na complementação) *As linguagens livres de contexto não são fechadas em relação à operação de complementação.*

Justificativa Dadas duas linguagens L_1 e L_2 quaisquer, é sabido (Lei de De Morgan, ver Teorema 1.1) que:

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

Nesta igualdade estão envolvidas apenas as operações de intersecção, união e complementação. Sabe-se, conforme demonstração anterior (Teorema 4.18), que as linguagens livres de contexto são fechadas em relação à operação de união. Se elas fossem fechadas também em relação à operação de complemento, deveriam necessariamente ser fechadas em relação à operação de intersecção. Como isso não é verdade (Teorema 4.21), conclui-se que as linguagens livres de contexto não são fechadas em relação à operação de complementação. ■

4.14 Questões Decidíveis e Não-Decidíveis

Assim como acontece com a classe das linguagens regulares, existem diversas questões acerca das linguagens livres de contexto que podem sempre ser decididas, quaisquer que sejam as linguagens envolvidas. Por outro lado, certas questões que são decidíveis para a classe das linguagens regulares não podem ser decididas no caso geral, quando transpostas para a classe das linguagens livres de contexto (por exemplo, a questão $L = \Sigma^*$ é decidível no caso das linguagens regulares, porém não é decidível no caso das linguagens livres de contexto).

Teorema 4.24 (A cadeia pertence?) *Sejam L uma linguagem livre de contexto sobre Σ e α uma cadeia pertencente a Σ^*. Então, a questão "$\alpha \in L$?" é decidível.*

Justificativa Conforme o Algoritmo 4.15.

Algoritmo 4.15 (A cadeia pertence?) *Determinação da pertinência da cadeia $w \in \Sigma^*$ à linguagem livre de contexto $L \subseteq \Sigma^*$.*

- Entrada: uma cadeia $w \in \Sigma^*$ e uma linguagem livre de contexto $L \subseteq \Sigma^*$;
- Saída: SIM, se $w \in \Sigma^*$; NÃO, caso contrário;
- Método:

  1. Obter uma gramática livre de contexto G tal que $L = L(G)$;

2. Se $w = \epsilon$, então:

   a) Determinar, conforme o Algoritmo 4.3 (eliminação de produções em vazio em gramáticas livres de contexto), se S (a raiz de G) pertence ao conjunto E; em caso afirmativo, a resposta é SIM; caso contrário, a resposta é NÃO;

3. Se $w \neq \epsilon$, então:

   a) Obter G' na Forma Normal de Greibach (ver Algoritmo 4.8 na Seção 4.6), tal que $L(G') = L(G) - \{\epsilon\}$;

   b) Considerar $n = \big|w\big|$;

   c) Considerar m como o maior número de produções definido para um não-terminal, entre todos os não-terminais de G';

   d) Obter todas as seqüências de derivações mais à esquerda que geram formas sentenciais cujo prefixo seja uma cadeia de terminais de comprimento máximo n (existem no máximo m^n seqüências distintas);

   e) Verificar se alguma dessas seqüências de derivação corresponde à geração da cadeia w; em caso afirmativo, a resposta é SIM; caso contrário, a resposta é NÃO.

■

Exemplo 4.45 Considere-se a gramática abaixo, já apresentada na Forma Normal de Greibach:

$$\begin{aligned} G &= (\{S, B, C, a, b, c\}, \{a, b, c\}, P, S) \\ P &= \{S \rightarrow aBC \mid bBC, \\ &\quad B \rightarrow bB \mid b, \\ &\quad C \rightarrow c\} \end{aligned}$$

Considere-se a cadeia $abbc \in L(G)$. Então, $n = 4$ e $m = 2$ (pois existem duas produções para S, duas para B e apenas uma para C) e, conforme o Algoritmo 4.15, existem no máximo $2^4 = 16$ seqüências distintas de derivações mais à esquerda que geram como prefixo uma cadeia de terminais de comprimento máximo 4, não havendo necessidade de se inspecionar outras seqüências. A Tabela 4.5 relaciona todas as oito seqüências que geram cadeias de terminais de comprimento máximo 4.

Seqüência			ΣV^*		$\Sigma\Sigma V^*$		$\Sigma\Sigma\Sigma V^*$		$\Sigma\Sigma\Sigma\Sigma V^*$
1	S	$\Rightarrow$	aBC	$\Rightarrow$	$abBC$	$\Rightarrow$	$abbBC$	$\Rightarrow$	$abbbBC$
2								$\Rightarrow$	$abbbC$
3						$\Rightarrow$	$abbC$	$\Rightarrow$	$abbc$
4				$\Rightarrow$	abC	$\Rightarrow$	abc		
5		$\Rightarrow$	bBC	$\Rightarrow$	$bbBC$	$\Rightarrow$	$bbbBC$	$\Rightarrow$	$bbbbBC$
6								$\Rightarrow$	$bbbbC$
7						$\Rightarrow$	$bbbC$	$\Rightarrow$	$bbbc$
8				$\Rightarrow$	bbC	$\Rightarrow$	bbc		

Tabela 4.5: Derivações mais à esquerda que geram prefixos de comprimento máximo 4

Como se pode verificar, a seqüência de derivações 3 produz a cadeia $abbc$. Por outro lado, a cadeia $bcbc$, também de comprimento 4, não pertence à linguagem, uma vez que nenhuma das oito seqüências da Tabela 4.5 gera o prefixo $bcbc$ e, portanto, nenhuma seqüência de derivações é capaz de gerar a cadeia $bcbc$. □

O tempo de execução do Algoritmo 4.15 é proporcional a m^n, onde n é o comprimento da cadeia de entrada. Logo, esse tempo varia exponencialmente com o tamanho da cadeia, o que é um resultado considerado ineficiente, e portanto indesejável do ponto de vista prático. Diversos outros algoritmos, no entanto, apresentam tempos de execução que são proporcionais a n^3 (ou menos), o que implica importantes ganhos de desempenho. Uma discussão sobre tais algoritmos, assim como referências, pode ser encontrada em [46], ou na literatura sobre compiladores, e foge ao escopo da presente publicação.

Por outro lado, conforme o Algoritmo 4.11 (obtenção de um autômato de pilha a partir de uma gramática livre de contexto na Forma Normal de Greibach), é possível construir um autômato de pilha que sempre pára, qualquer que seja a cadeia de entrada que lhe seja submetida. De fato, cada movimentação do autômato construído com base naquele algoritmo efetua o consumo de um símbolo da cadeia de entrada. Como o comprimento dessa cadeia é finito, e inexistem ciclos formados por transições em vazio em tal autômato, seu processamento pára após um número finito de passos (correspondente ao comprimento da cadeia de entrada), e o mesmo pode ser usado como alternativa para se determinar se uma cadeia qualquer (não-vazia) pertence a uma linguagem livre de contexto.

Fica, portanto, demonstrada a existência de autômatos de pilha que sempre param, quaisquer que sejam a linguagem livre de contexto considerada e a cadeia de entrada que lhes sejam submetidas. Essa propriedade é análoga à obtida no Teorema 3.26 (pertinência de uma cadeia a uma linguagem regular), que garante a existência de autômatos finitos que sempre param, quaisquer que sejam a linguagem regular considerada e a cadeia de entrada que lhes seja submetida.

Teorema 4.25 (A linguagem é vazia?) *Seja L uma linguagem livre de contexto. Então, a questão "$L = \varnothing$?" é decidível.*

Justificativa $L = \varnothing$ se e somente se L não contém nenhuma sentença de comprimento menor que n, onde n é a constante definida pelo "Pumping Lemma" para linguagens livres de contexto. Tal fato pode ser verificado por demonstração semelhante à que foi feita no Teorema 3.23 (determina se uma linguagem regular é vazia ou não-vazia).

Uma maneira alternativa de se verificar se $L = \varnothing$ consiste em obter uma gramática $G = (V, \Sigma, P, S)$, sem símbolos inúteis, que gere L. Se $S \in V$, então L é não-vazia. Caso contrário, L é vazia. ■

Teorema 4.26 (A linguagem é infinita?) *Seja L uma linguagem livre de contexto. Então, a questão "L é infinita?" é decidível.*

Justificativa L é infinita se e somente se contiver pelo menos uma sentença de comprimento maior ou igual a n e menor que $2n$, onde n é a constante definida pelo "Pumping Lemma" para linguagens livres de contexto. Tal fato pode ser verificado por demonstração semelhante à que foi feita no Teorema 3.24 (determina se uma linguagem regular é finita ou infinita).

Uma outra maneira de se determinar se L é infinita consiste em obter uma gramática sem símbolos inúteis que gere L, e depois verificar se existem não-terminais X auto-recursivos na mesma ($X \Rightarrow \alpha X \beta, \alpha\beta \in \Sigma^+$). Caso exista pelo menos um não-terminal auto-recursivo (não necessariamente central), então L é infinita. Caso contrário, L é finita. ■

Teorema 4.27 (A linguagem é finita?) *Seja L uma linguagem livre de contexto. Então, a questão "L é finita?" é decidível.*

Justificativa Decorre diretamente do resultado dos teoremas anteriores: se L não é infinita (Teorema 4.26), então L é finita. Se $L = \varnothing$ (Teorema 4.25), então L é finita e vazia, caso contrário é finita e não-vazia. ■

As seguintes questões acerca das linguagens e das gramáticas livres de contexto (respectivamente L, G, G_1 e G_2 quaisquer) não são decidíveis e não serão demonstradas neste livro.

- G é ambígua?
- L é inerentemente ambígua?
- $L(G)$ é regular?
- $L(G) = \Sigma^*$?
- $L(G_1) = L(G_2)$?
- $L(G_1) \subseteq L(G_2)$?
- $L(G_1) \cap L(G_2) = \varnothing$?
- $L(G_1) \cap L(G_2)$ é livre de contexto?
- $\overline{L(G_1)}$ é livre de contexto?

Naturalmente, por se tratar de questões não-decidíveis, demonstra-se a inexistência de quaisquer algoritmos que solucionem essas questões no caso geral. Tais demonstrações podem ser encontradas em [46].

5 Linguagens Sensíveis ao Contexto

Linguagens sensíveis ao contexto são aquelas cujas sentenças exibem características de dependência — ou vinculação — entre trechos distintos das mesmas. Ou seja, determinadas partes de uma sentença só serão consideradas válidas se ocorrerem simultaneamente a trechos relacionados, presentes em outras regiões da mesma sentença. Daí a origem do nome "sensibilidade ao contexto".

Deve-se, no entanto, entender corretamente o significado do termo "sensibilidade ao contexto", também conhecido como "dependência de contexto", pois não é qualquer tipo de vinculação que caracteriza esta propriedade das linguagens.

Em particular, não se está interessado em quaisquer das dependências que possam ser representadas por gramáticas livres de contexto, tais como seqüências simples de símbolos ou mesmo o balanceamento de símbolos, típico das linguagens livres de contexto.

Exemplo 5.1 Seja a linguagem das sentenças que representam expressões aritméticas com até quatro operações sobre o alfabeto $\{a, b\}$, gerada pela gramática:

$$
\begin{array}{rcl}
E & \rightarrow & E \text{ “}+\text{” } E \\
 & & \mid E \text{ “}*\text{” } E \\
 & & \mid E \text{ “}-\text{” } E \\
 & & \mid E \text{ “}/\text{” } E \\
 & & \mid \text{“(”} E \text{“)”} \\
 & & \mid \text{“}a\text{”} \\
 & & \mid \text{“}b\text{”}
\end{array}
$$

Um exemplo de sentença pertencente a esta linguagem é $a - (b + (a/b) * a)$. É claro que, neste caso, é possível verificar algumas dependências de contexto, no sentido literal da palavra. Por exemplo, as ocorrências dos parênteses. Não seria possível fechar o segundo parênteses se o primeiro não tivesse sido aberto. Outro exemplo é que, de cada lado do símbolo de divisão, deve existir uma letra "a" ou "b", representando cada um dos operandos desta operação. □

Tais tipos de "dependência de contexto" podem ser facilmente representados por gramáticas livres de contexto e, por isso, não serão considerados como tal neste estudo.

Aqui, por outro lado, o interesse é maior em outros tipos de dependências contextuais, que não possam ser representadas através das gramáticas livres de contexto. Tais dependências caracterizam linguagens mais complexas que as linguagens livres de contexto, e constituem o objeto deste capítulo.

É preciso, portanto, analisar esse tipo de dependências, referidas nesta introdução. Um bom exemplo para o seu entendimento são as dependências de contexto que são encontradas nas linguagens de programação de alto nível mais comuns.

Tais linguagens costumam oferecer declarações diversas, como, por exemplo, declarações de tipos, de constantes, de variáveis, de procedimentos, de funções etc. Tais declarações cumprem o propósito duplo de acrescentar novos nomes ao espaço de nomes criado pelo programador, e de associar-lhes atributos (por exemplo, no caso das variáveis, seus tipos; no caso das funções, a quantidade e o tipo de seus parâmetros, bem como o tipo do valor retornado).

Uma vez declarados, os novos nomes podem ser empregados no corpo do programa. Normalmente, tais linguagens exigem que apenas nomes declarados possam ser referenciados em comandos e expressões; além disso, que a utilização dos mesmos seja feita de forma coerente com os atributos que lhes foram associados.

É o caso, por exemplo, de se utilizar, no lado esquerdo de um comando de atribuição, apenas variáveis declaradas e visíveis no local da referência. É o caso, também, de se utilizar, na expressão do lado direito de um comando de atribuição, apenas os nomes e literais cujos atributos sejam compatíveis com as operações que estejam sendo empregadas.

Dessa forma, a correção sintática de uma sentença, no caso um programa escrito em uma linguagem de alto nível, só se completa se as condições anteriores forem verificadas. Está criada, portanto, uma dependência de contexto entre os atributos dos nomes constantes nas respectivas declarações e a forma como os mesmos são manipulados ao longo do programa.

Exemplo 5.2 Considere-se, por exemplo, o seguinte trecho de programa escrito na linguagem C:

```
void main () { int x; scanf (“%d”,&x); printf (“%d”,x);}
```

Existe uma clara vinculação entre a declaração da variável "x" e a referência à mesma nos comandos de leitura e impressão. Se ela não tivesse sido declarada, ou se os seus atributos não fossem compatíveis com a maneira como ela é utilizada nesses comandos, não seria possível considerar tal sentença como válida do ponto de vista sintático. □

Dependências de contexto são, portanto, uma característica fundamental das linguagens de programação em geral. Do ponto de vista mais teórico, as linguagens formais com tais características constituem uma classe de linguagens sobre a qual recaem interesses e questões similares às estudadas nos capítulos anteriores, entre as quais, maneiras de formalizá-las do ponto de vista gramatical, modelos de reconhecimento, propriedades mais importantes e seu relacionamento com as demais classes de linguagens.

5.1 Gramáticas Sensíveis ao Contexto

Cabe, neste ponto, justificar a incapacidade das gramáticas livres de contexto para representar linguagens sensíveis ao contexto. Rigorosamente, o "Pumping Lemma" para linguagens livres de contexto, apresentado no Capítulo 4 (Teorema 4.17), cumpre o papel de demonstrar a existência de linguagens que não sejam livres de contexto, ou seja, de linguagens que não podem ser representadas através de gramáticas livres de contexto. Uma análise um pouco mais detalhada da natureza das gramáticas livres de contexto revela a origem das suas limitações e serve como elemento motivador para a conceituação de uma nova classe de gramáticas — a das gramáticas sensíveis ao contexto.

Ao restringir o lado esquerdo das regras gramaticais a um único símbolo não-terminal, as gramáticas livres de contexto estabelecem que todas as derivações sejam feitas considerando-se apenas o não-terminal selecionado para a substituição, não importando os símbolos (terminais e não-terminais) que o rodeiam à esquerda e à direita — ou seja, o seu contexto. Daí a origem do termo "livre de contexto", pois a estrutura sintática dos trechos já desenvolvidos da cadeia nunca é levada em consideração quando da substituição de um não-terminal pelo lado direito da regra escolhida.

Isso torna impossível representar casos como o da declaração de variáveis, uma vez que a substituição dos símbolos não-terminais que representam, por exemplo, a declaração de variável e o comando de atribuição não podem ser vinculados durante o processo de derivação da cadeia, para evitar, por exemplo, a geração de sentenças que contenham referências a nomes inválidos ou de tipos incompatíveis com a operação aplicada.

Para superar tal limitação, empregam-se as gramáticas sensíveis ao contexto. Formalmente, uma **gramática sensível ao contexto** $G = (V, \Sigma, P, S)$ é aquela cujas regras do conjunto P obedecem ao formato $\alpha \rightarrow \beta$, onde:

- $\alpha \in V^*NV^*$
- $\beta \in V^*$
- $|\beta| \geq |\alpha|$

Gramáticas sensíveis ao contexto eliminam a restrição de que o lado esquerdo das regras seja formado por um único símbolo, e de que este seja um símbolo não-terminal. Tais gramáticas admitem qualquer quantidade de símbolos do lado esquerdo, sejam eles terminais ou não-terminais. Exigem, apenas, que do lado esquerdo exista pelo menos um símbolo não-terminal e, também, que o lado direito possua uma quantidade de símbolos não inferior àquela encontrada no lado esquerdo da mesma regra.

Define-se inicialmente **linguagem sensível ao contexto** como sendo aquela que possa ser definida através de uma gramática sensível ao contexto. Conforme convencionado na Seção 2.6, essa definição é estendida para qualquer linguagem L que contenha a cadeia vazia, desde que $L - \{\epsilon\}$ possa ser gerada por uma gramática sensível ao contexto.

Define-se **linguagem estritamente sensível ao contexto** como sendo uma linguagem sensível ao contexto mas não livre de contexto.

Exemplo 5.3 Seja a gramática $G_1 = (\{a, b, c, S, B, C\}, \{a, b, c\}, P, S)$, com

$$\begin{aligned} P \quad = \quad & \{S \rightarrow aSBC, \\ & S \rightarrow aBC, \\ & CB \rightarrow BC, \\ & aB \rightarrow ab, \\ & bB \rightarrow bb, \\ & bC \rightarrow bc, \\ & cC \rightarrow cc\} \end{aligned}$$

Todas as regras desta gramática satisfazem à condição de possuir pelo menos um não-terminal do lado esquerdo e uma seqüência arbitrária de símbolos do lado direito, porém de comprimento nunca inferior ao comprimento verificado no lado esquerdo da mesma regra.

Algumas derivações possíveis são:

- $S \Rightarrow aBC \Rightarrow abC \Rightarrow abc$;
- $S \Rightarrow aSBC \Rightarrow aaBCBC \Rightarrow aaBBCC \Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbcC \Rightarrow aabbcc$;
- $S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaBCBCBC \Rightarrow aaaBBCCBC \Rightarrow aaaBBCBCC \Rightarrow aaaBBBCCC \Rightarrow aaabBBCCC \Rightarrow aaabbBCCC \Rightarrow aaabbbCCC \Rightarrow aaabbbcCC \Rightarrow aaabbbccC \Rightarrow aaabbbccc$.

É fácil demonstrar que $L(G_1) = \{a^n b^n c^n \mid n \geq 1\}$. É também possível demonstrar, pela aplicação do "Pumping Lemma" para linguagens livres de contexto, que $L(G_1)$ não é uma linguagem livre de contexto (Exemplo 4.38). Logo, não existe qualquer gramática livre de contexto que seja capaz de gerar $L(G_1)$, e portanto L é uma linguagem estritamente sensível ao contexto.

Uma rápida análise das produções de G_1 possibilita a compreensão da lógica que existe em sua concepção: as duas regras iniciais geram formas sentenciais nas quais uma certa quantidade de símbolos "a" é seguida por pares "BC", de tal forma que as quantidades de "a", "B" e "C" sejam idênticas — como em $aaaBCBCBC$. A regra $CB \rightarrow BC$ permite que os "B"s e os "C"s sejam agrupados — gerando, neste exemplo, $aaaBBBCCC$. As demais regras efetuam as substituições dos não-terminais pelos terminais correspondentes, com o cuidado de evitar que tais substituições possam ser utilizadas para gerar sentenças distintas das que estão relacionadas acima. □

As gramáticas sensíveis ao contexto são ditas **monotônicas**, pois o comprimento das formas sentenciais obtidas durante o processo de derivação de uma sentença nunca sofre redução. Esta característica será explorada na Seção 5.5, para fins de associação desta classe de linguagens com o seu respectivo modelo de reconhecimento mais simples.

Exemplo 5.4 Considere-se a gramática $G_2 = (\{a, b, c, S, A, B, C\}, \{a, b, c\}, P, S)$, com

$$\begin{aligned} P \quad = \quad & \{S \rightarrow ABC, \\ & S \rightarrow ABCS, \\ & AB \rightarrow BA, \\ & AC \rightarrow CA, \\ & BA \rightarrow AB, \\ & BC \rightarrow CB, \\ & CA \rightarrow AC, \\ & CB \rightarrow BC, \\ & A \rightarrow a, \\ & B \rightarrow b, \\ & C \rightarrow c\} \end{aligned}$$

Analise-se agora a derivação de algumas sentenças:

- $S \Rightarrow ABC \Rightarrow aBC \Rightarrow abC \Rightarrow abc$;
- $S \Rightarrow ABC \Rightarrow ACB \Rightarrow CAB \Rightarrow CBA \Rightarrow cBA \Rightarrow cbA \Rightarrow cba$;
- $S \Rightarrow ABCS \Rightarrow ABCABC \Rightarrow BACABC \Rightarrow BACBAC \Rightarrow BACBCA \Rightarrow bACBCA \Rightarrow baCBCA \Rightarrow bacBCA \Rightarrow bacbCA \Rightarrow bacbcA \Rightarrow bacbca$.

As duas regras iniciais inserem quantidades idênticas de símbolos A, B e C na forma sentencial. As regras seguintes são utilizadas para "embaralhar" esses símbolos de maneira arbitrária. As três últimas substituem cada símbolo não-terminal pelo respectivo símbolo terminal.

A linguagem gerada por G_2 consiste em todas as sentenças sobre $\{a, b, c\}$ — com comprimento mínimo 3 —, de tal forma que as quantidades desses símbolos sejam sempre idênticas. Formalmente:

$$L(G_2) = \{w \in \Sigma^* \mid \text{ as quantidades de "}a\text{", "}b\text{" e "}c\text{" em } w \text{ são idênticas, e } \left| w \right| \geq 3\}$$

Como no Exemplo 5.3, pode-se demonstrar que esta linguagem é estritamente sensível ao contexto. Para isso é suficiente provar, através do "Pumping Lemma", que ela não é livre de contexto□

Gramáticas deste tipo geram linguagens denominadas sensíveis ao contexto, ou simplesmente do tipo 1. A inspeção rigorosa dos formatos admitidos para as produções das gramáticas do tipo 1 não permite a caracterização imediata de toda e qualquer gramática (e conseqüentemente das correspondentes linguagens) do tipo 2 como sendo também do tipo 1. Isso decorre do fato de que as gramáticas do tipo 2 admitem a cadeia vazia ϵ como alternativa de substituição para o lado esquerdo de qualquer produção, o que não é permitido em gramáticas do tipo 1 em função da restrição $\left|\alpha\right| \leq \left|\beta\right|$.

Rigorosamente, uma linguagem é dita **sensível ao contexto** se e somente se:

- $\epsilon \notin L$ e $L = L(G)$, onde G é uma gramática sensível ao contexto, ou
- $\epsilon \in L$ e $L - \{\epsilon\}$ pode ser gerada por uma gramática sensível ao contexto.

Neste último caso, aceita-se que a regra $S \rightarrow \epsilon$ seja incluída na gramática, porém desde que S (a raiz da gramática) não compareça do lado direito de nenhuma outra regra. Dessa forma evita-se a geração de formas sentenciais que possam sofrer contração

durante o processo de derivação. A única possibilidade de isso ocorrer corresponde à derivação inicial, quando o símbolo a ser substituído é a própria raiz da gramática.

Cumpre, neste ponto, estabelecer uma importante relação entre linguagens livres de contexto e linguagens sensíveis ao contexto. Suponha-se, inicialmente, que $L = L(G)$, onde G é uma gramática livre de contexto, e, adicionalmente, que $\epsilon \notin L$. É fácil perceber, neste caso, que G satisfaz a todas as especificações de uma gramática sensível ao contexto, pois não haverá nenhuma regra com ϵ à direita. Logo, $L(G)$ será também uma linguagem sensível ao contexto. Ou seja: linguagens livres de contexto que não contêm a cadeia vazia são também linguagens sensíveis ao contexto.

Caso $\epsilon \in L(G)$, e G seja uma gramática livre de contexto, será necessário aplicar transformações em G obtendo-se G', de modo que:

- $S \rightarrow \epsilon$ seja a única regra vazia em G';
- S não compareça no lado direito de nenhuma outra regra de G';
- $L(G) = L(G')$.

Como foi demonstrado no Teorema 4.7, tal transformação é sempre possível. Toda e qualquer gramática livre de contexto pode ser transformada, sem prejuízo da linguagem que estiver sendo definida, em uma outra gramática em que a cadeia vazia comparece apenas na produção $S \rightarrow \epsilon$, sendo S a raiz da gramática. Além disso, S garantidamente não figura no lado direito de nenhuma outra produção. Isso significa que o único efeito prático da presença de produções do tipo $A \rightarrow \epsilon, A \in N$, em gramáticas livres de contexto quaisquer, é na eventual incorporação da cadeia vazia como sentença na linguagem definida.

Portanto, com exceção da produção $S \rightarrow \epsilon$, as gramáticas do tipo 2 podem ser sempre convertidas para um formato que as torne um caso particular das gramáticas do tipo 1. Em outras palavras, tem-se que qualquer gramática do tipo 2, desde que devidamente convertida para esse formato padronizado, e a menos da produção $S \rightarrow \epsilon$, torna-se também uma gramática do tipo 1.

Assim, é fato que G', exceto pela regra vazia, é uma gramática sensível ao contexto e, conseqüentemente, $L(G') - \{\epsilon\}$ é uma linguagem sensível ao contexto. Logo, $L(G) = L(G')$ é, de acordo com a definição, uma linguagem sensível ao contexto.

Como conseqüência, pode-se concluir que toda linguagem livre de contexto é também uma linguagem sensível ao contexto. Fica claro, também, que as linguagens livres de contexto constituem um subconjunto próprio das linguagens sensíveis ao contexto.

Para finalizar, algumas observações de ordem prática. É muito comum, quando se especificam linguagens de programação, fazer uso de gramáticas livres de contexto. Tal artifício mascara, no entanto, a existência das dependências de contexto, as quais acabam sendo excluídas da formalização gramatical. Esta estratégia possui pelo menos uma importante razão de ser e uma importante conseqüência:

- A razão de ser consiste no fato de que formalismos distintos, adequados para a representação de linguagens sensíveis ao contexto (como, por exemplo, as gramáticas sensíveis ao contexto), são relativamente complexos e difíceis de serem trabalhados quando se trata de especificar as dependências de contexto típicas das linguagens de programação. Opta-se, assim, por uma questão de praticidade, pelo uso de formalismos mais simples, em particular os livres de contexto.

- A conseqüência dessa prática é que a linguagem resultante será um superconjunto da linguagem pretendida. Além das sentenças que observam estritamente as regras de dependências de contexto da linguagem, outras cadeias, indesejadas, que não observam tais dependências, acabam sendo incorporadas à linguagem definida. E disso resulta a necessidade de se acrescentarem filtros para eliminá-las da linguagem, preservando-se apenas as cadeias efetivamente aderentes à especificação original.

Trata-se, portanto, de uma estratégia originada a partir de uma necessidade de simplificar a solução do problema, mas que resulta, em momento posterior, na necessidade de incorporação de filtros que descartem as cadeias inválidas.

Exemplo 5.5 Considere-se a gramática definida pelo conjunto de produções a seguir, representativa de uma linguagem de programação que possibilita declarações de variáveis seguidas de comandos de atribuição e expressões.

$$
\begin{array}{rcl}
\{Programa & \rightarrow & Declaracoes\ Comandos, \\
Declaracoes & \rightarrow & Declaracoes\ Declaracao \\
 & & \mid \epsilon, \\
Declaracao & \rightarrow & \text{“}\%\text{”}\, Identificador, \\
Comandos & \rightarrow & Comandos\ Comando \\
 & & \mid \epsilon, \\
Comando & \rightarrow & \text{“}\#\text{”}\, Identificador \text{“}=\text{”}\, Expressao, \\
Expressao & \rightarrow & Expressao \text{“}+\text{”}\, Expressao \\
 & & \mid Expressao \text{“}*\text{”}\, Expressao \\
 & & \mid Identificador, \\
Identificador & \rightarrow & \text{“}a\text{”} \\
 & & \mid \text{“}b\text{”} \\
 & & \mid \text{“}c\text{”}\}
\end{array}
$$

Um exemplo de sentença pertencente a esta linguagem é:

$$
\begin{array}{l}
\%a \\
\%b \\
\#a = a + b \\
\#b = b * b
\end{array}
$$

Tal sentença pode ser entendida como um programa contendo a declaração de duas variáveis (“a” e “b”, nas duas linhas iniciais) e o uso de ambas em dois comandos de atribuição (nas duas linhas finais).

A única dependência de contexto exibida por essa linguagem é que os únicos identificadores que podem ser usados nos comandos (após o símbolo “#”) são aqueles que tenham sido declarados anteriormente (após o símbolo “%”).

Perceba-se, no entanto, que tal regra não está incorporada na gramática apresentada — uma gramática livre de contexto —, mas é bastante simples e fácil de ser interpretada.

Tal facilidade de representação, entretanto, é onerada pela possibilidade de derivação de sentenças como, por exemplo:

$$
\begin{array}{l}
\%a \\
\%c \\
\#a = a + b \\
\#b = b * b
\end{array}
$$

que viola a dependência de contexto que vincula referências às variáveis com as respectivas declarações (a variável “b” é referenciada indevidamente, uma vez que sua declaração foi substituída pela declaração da variável “c”). Daí a necessidade de se adicionar um filtro para excluir tais sentenças do conjunto gerado pela gramática apresentada. □

5.2 Gramáticas com Derivações Controladas

Este item é novo no texto.
Se, por um lado, as gramáticas livres de contexto não possuem poder suficiente para representar linguagens mais complexas, como é o caso das linguagens sensíveis ao contexto (conforme demonstrado na seção 4.10), por outro é curioso perceber como o uso de alguns mecanismos simples de extensão as tornam capazes de lidar supreendentemente bem com essa complexidade, superando as limitações que lhe são intrínsecas.

Tais mecanismos, genericamente denominados "controladores de derivações", são extensões gramaticais que restringem — ou controlam —, de alguma forma, a escolha das regras de produção possíveis de serem aplicadas às formas sentenciais geradas por uma gramática subjacente.

Em gramáticas convencionais, de qualquer tipo, a escolha da regra de produção a ser aplicada em uma forma sentencial γ qualquer é resultado, única e exclusivamente, da identificação, em γ, de alguma subcadeia α que coincida com o lado esquerdo de alguma regra $p \in P$ (seção 2.3). Eventualmente esse critério resulta num conjunto de regras que satisfazem tal condição, dentre as quais a escolha de uma determinada regra é livre e válida, podendo o seu uso conduzir, ou não, à geração de alguma sentença.

Gramáticas com derivações controladas vão além dessa exigência básica, restringindo, de forma complementar e através de mecanismos diversos, o conjunto das regras que normalmente seriam aplicáveis a cada forma sentencial durante a geração de uma sentença. Como resultado consegue-se gerenciar melhor o uso das regras de produção, e assim estender o poder de representação lingüística da gramática subjacente.

Os formalismos estudados a seguir possuem gramáticas livres de contexto como dispositivo subjacente. Nos exemplos apresentados, eles são empregados para representar linguagens estritamente sensíveis ao contexto. Não obstante, dispositivos gramaticais subjacentes dos tipos 3, 2, 1 ou 0 podem ser usados livremente com qualquer um deles.

A demonstração da equipotência entre os formalismos a seguir apresentados, e destes com as gramáticas do tipo 0 (irrestritas), pode ser encontrada em [37], [38] ou [41]. Essa condição, no entanto, só é alcançada quando eles são estendidos por um mecanismo de derivação complementar, denominado "verificação de aparência", que não será discutido aqui e cuja definição pode ser encontrada nessas mesmas referências.

A seguir são apresentados os principais formalismos gramaticais que fazem uso de mecanismos controladores de derivações:

- Gramáticas controladas por linguagens regulares;
- Gramáticas matriciais;
- Gramáticas programadas;
- Gramáticas periodicamente variantes no tempo.

A diversidade de dispositivos gramaticais com tais características é grande, como pode ser verificado em [39], que apresenta 25 deles. Em [37], [38] e [41] são encontrados bons textos sobre gramáticas com derivações controladas, assim como uma vasta gama de resultados teóricos acerca das classes de linguagens por elas geradas. Os exemplos mostrados a seguir foram inspirados e/ou adaptados de exemplos extraídos dessas referências. Com o objetivo de facilitar a comparação entre os formalismos, especialmente no que se refere ao modelo de funcionamento dos mesmos, cada um deles é exemplifi-

cado com as correspondentes gramáticas para as mesmas duas linguagens estritamente sensíveis ao contexto:

- $\{a^n b^n c^n \mid n \geq 1\}$
- $\{ww \mid w \in \{a, b\}^+\}$

A notação apresentada a seguir difere da utilizada na literatura referenciada, e foi elaborada com o objetivo de (*i*) padronizar, tanto quanto possível, a representação algébrica dos formalismos considerados; (*ii*) evidenciar a gramática subjacente tradicional, presente em cada formalismo (nos casos apresentados, gramáticas livres de contexto) e (*ii*) permitir a análise comparativa dos mecanismos de extensão que caracterizam cada formalismo.

Gramáticas controladas por linguagens regulares

São aquelas em que uma gramática subjacente $G' = (V, \Sigma, P, S)$ é complementada por uma expressão regular R definida sobre o seu conjunto de produções P. Uma seqüência de derivações é considerada válida se e apenas se a seqüência das regras utilizadas nesse processo corresponder a algum elemento de R. A noção de uso de uma linguagem de controle foi apresentada pela primeira vez em [42]. Uma gramática livre de contexto controlada por linguagem regular G é definida formalmente como:

$$\begin{aligned}
G &= (G', R) \\
G' &= (V, \Sigma, P, S) \text{ é uma gramática livre de contexto} \\
P &= \{p_1, p_2, ..., p_n\} \\
R &= \text{é uma expressão regular sobre } P \\
L(G) &= \{w \in \Sigma^* \mid \\
&\quad (S \Rightarrow_{p_{i_1}} w_1 \Rightarrow_{p_{i_2}} ... \Rightarrow_{p_{i_k}} w_k = w) \text{ e} \\
&\quad (p_{i_1}, p_{i_2}, ..., p_{i_k} \in R)\}
\end{aligned}$$

Exemplo 5.6

$$\begin{aligned}
G_1 &= (G_1', R) \\
G_1' &= (\{S, B, C, a, b, c\}, \{a, b, c\}, \{p_1, p_2, p_3, p_4, p_5\}, S) \\
(p_1) &\quad S \rightarrow BC \\
(p_2) &\quad B \rightarrow aBb \\
(p_3) &\quad B \rightarrow ab \\
(p_4) &\quad C \rightarrow cC \\
(p_5) &\quad C \rightarrow c \\
R &= p_1(p_2p_4)^*(p_3p_5) \\
L(G_1) &= \{a^n b^n c^n \mid n \geq 1\}
\end{aligned}$$

Considere-se a cadeia $p_1p_2p_4p_2p_4p_3p_5 \in R$. A aplicação do correspondente conjunto de produções gera a sentença *aaabbbccc*, conforme mostra a seqüência de derivações:

$$\begin{aligned}
&S \Rightarrow_{p_1} BC \Rightarrow_{p_2} aBbC \Rightarrow_{p_4} aBbcC \Rightarrow_{p_2} \\
&aaBbbcC \Rightarrow_{p_4} aaBbbccC \Rightarrow_{p_3} aaabbbccC \Rightarrow_{p_4} aaabbbccc
\end{aligned}$$

□

Exemplo 5.7

$$
\begin{array}{rcl}
G_2 & = & (G_2', R) \\
G_2' & = & (\{S, A, B, a, b\}, \{a, b\}, \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}) \\
(p_1) & & S \rightarrow AB \\
(p_2) & & A \rightarrow aA \\
(p_3) & & B \rightarrow aB \\
(p_4) & & A \rightarrow bA \\
(p_5) & & B \rightarrow bB \\
(p_6) & & A \rightarrow a \\
(p_7) & & B \rightarrow a \\
(p_8) & & A \rightarrow b \\
(p_9) & & B \rightarrow b \\
R & = & p_1(p_2p_3 \mid p_4p_5)^*(p_6p_7 \mid p_8p_9) \\
L(G_2) & = & \{ww \mid w \in \{a, b\}^+\}
\end{array}
$$

Considere-se a cadeia $p_1p_2p_3p_4p_5p_8p_9 \in R$. A aplicação do correspondente conjunto de produções gera a sentença $abbabb$, conforme mostra a seqüência de derivações:

$$
\begin{array}{l}
S \Rightarrow_{p_1} AB \Rightarrow_{p_2} aAB \Rightarrow_{p_3} aAaB \Rightarrow_{p_4} \\
abAaB \Rightarrow_{p_5} abAabB \Rightarrow_{p_8} abbabB \Rightarrow_{p_9} abbabb
\end{array}
$$

□

Gramáticas matriciais

Correspondem a gramáticas $G = (V, \Sigma, P, S)$ que são complementadas por uma estrutura matricial M, na qual estão distribuídas as regras de produção de P. Uma seqüência de derivações é considerada válida se e apenas se as regras forem sempre aplicadas em grupos, cada grupo sendo formado por todas as regras de uma mesma linha, da primeira à última coluna, nessa seqüência. Gramáticas matriciais foram inicialmente divulgadas em [43]. Uma gramática livre de contexto matricial G é definida formalmente como:

$$
\begin{array}{rcl}
G & = & (G', M) \\
G' & = & (V, \Sigma, P, S) \text{ é uma gramática livre de contexto} \\
M & = & \{m_1, m_2, ..., m_n\} \\
m_i & = & (p_{i_1}, ..., p_{i_k}) \mid k \geq 1, p_{i_j} \in P, i_1 \leq i_j \leq i_k \\
L(G) & = & \{w \in \Sigma^* \mid \\
 & & (S = w_0 \Rightarrow_{m_{i_1}} w_1 \Rightarrow_{m_{i_2}} w_2 \Rightarrow_{m_{i_3}} ... \Rightarrow_{m_{i_{q-1}}} w_{q-1} \Rightarrow_{m_{i_q}} w_q = w), \text{ com} \\
 & & (m_{i_1}, m_{i_2}, m_{i_3}, ..., m_{i_{q-1}}, m_{i_q} \in M) \text{ e} \\
 & & (w_j \Rightarrow_{m_i} w_{j+1} : w_j \Rightarrow_{p_{i_1}} w_{j_1} \Rightarrow_{p_{i_2}} w_{j_2} \Rightarrow_{p_{i_3}} ... \Rightarrow_{p_{i_k}} w_{j_k} = w_{j+1})\}
\end{array}
$$

Exemplo 5.8

$$
\begin{array}{rcl}
G_3 & = & (G_3', M\}) \\
G_3' & = & (\{S, A, B, C, a, b, c\}, \{a, b, c\}, P, S) \\
P & = & \{S \rightarrow abc, S \rightarrow aAbBcC, A \rightarrow aA, B \rightarrow bB, C \rightarrow cC, A \rightarrow a, B \rightarrow b, C \rightarrow c\}
\end{array}
$$

$$
\begin{aligned}
M &= \{m_1, m_2, m_3, m_4\} \\
m_1 &= (S \rightarrow abc) \\
m_2 &= (S \rightarrow aAbBcC) \\
m_3 &= (A \rightarrow aA, B \rightarrow bB, C \rightarrow cC) \\
m_4 &= (A \rightarrow a, B \rightarrow b, C \rightarrow c) \\
L(G_3) &= \{a^n b^n c^n \mid n \geq 1\}
\end{aligned}
$$

Considere-se a seqüência $m_2 m_3 m_4$. A aplicação do correspondente conjunto de produções gera a sentença $aaabbbccc$, conforme mostra a seqüência de derivações:

$$S \Rightarrow_{m_2} aAbBcC \Rightarrow_{m_3} aaAbbBccC \Rightarrow_{m_4} aaabbbccc$$

As derivações parciais, implícitas na seqüência acima, são:

$$
\begin{aligned}
S \Rightarrow_{m_2} aAbBcC &: S \Rightarrow aAbBcC \\
aAbBcC \Rightarrow_{m_3} aaAbbBccC &: aAbBcC \Rightarrow aaAbBcC \Rightarrow aaAbbBcC \Rightarrow aaAbbBccC \\
aaAbbBccC \Rightarrow_{m_4} aaabbbccc &: aaAbbBccC \Rightarrow aaabbBccC \Rightarrow aaabbbccC \Rightarrow aaabbbccc
\end{aligned}
$$

□

Exemplo 5.9

$$
\begin{aligned}
G_4 &= (G'_4, M\}) \\
G'_4 &= (\{S, A, B, a, b\}, \{a, b\}, P, S) \\
P &= \{S \rightarrow AB, A \rightarrow aA, B \rightarrow aB, A \rightarrow bA, B \rightarrow bB, A \rightarrow a, B \rightarrow a, A \rightarrow b, B \rightarrow b\} \\
M &= \{m_1, m_2, m_3, m_4, m_5\}) \\
m_1 &= (S \rightarrow AB) \\
m_2 &= (A \rightarrow aA, B \rightarrow aB) \\
m_3 &= (A \rightarrow bA, B \rightarrow bB) \\
m_4 &= (A \rightarrow a, B \rightarrow a) \\
m_5 &= (A \rightarrow b, B \rightarrow b) \\
L(G_4) &= \{ww \mid w \in \{a, b\}^+\}
\end{aligned}
$$

Considere-se a seqüência $m_1 m_2 m_3 m_5$. A aplicação do correspondente conjunto de produções gera a sentença $aaabbbccc$, conforme mostra a seqüência de derivações:

$$S \Rightarrow_{m_1} AB \Rightarrow_{m_2} aAaB \Rightarrow_{m_3} abAabB \Rightarrow_{m_5} abbabb$$

As derivações parciais, implícitas na seqüência acima, são:

$$
\begin{aligned}
S \Rightarrow_{m_1} AB &: S \Rightarrow AB \\
AB \Rightarrow_{m_2} aAaB &: AB \Rightarrow aAB \Rightarrow aAaB \\
aAaB \Rightarrow_{m_3} abAabB &: aAaB \Rightarrow abAaB \Rightarrow abAabB \\
abAabB \Rightarrow_{m_5} abbabb &: abAabB \Rightarrow abbabB \Rightarrow abbabb
\end{aligned}
$$

□

Gramáticas programadas

Correspondem a gramáticas $G = (V, \Sigma, P, S)$ que são complementadas por uma estrutura T, na qual cada regra $p \in P$ é associada a um conjunto, eventualmente vazio, de regras de P. Uma seqüência de derivações é considerada válida se e apenas se, para cada regra p aplicada, a regra seguinte, a ser usada na próxima derivação, for sempre obtida no conjunto de regras associadas a p. Gramáticas programadas foram originalmente repor-

tadas em [44]. Uma gramática livre de contexto programada G é definida formalmente como:

$$\begin{aligned}
G &= (G', T) \\
G' &= (V, \Sigma, P, S) \text{ é uma gramática livre de contexto} \\
T &= \{t_1, t_2, ..., t_n\} \\
t_i &= (p_i, s_i) \mid 1 \leq i \leq n, p_i \in P, s_i \in 2^T \\
L(G) &= \{w \in \Sigma^* \mid \\
&\quad (S \Rightarrow_{p_{i_1}} w_1 \Rightarrow_{p_{i_2}} ... \Rightarrow_{p_{i_q}} w) \text{ e} \\
&\quad (p_{i_{j+1}} \in s_{i_j}, j \geq 1)\}
\end{aligned}$$

Exemplo 5.10

$$\begin{aligned}
G_5 &= (G'_5, T\}) \\
G'_5 &= (\{S, A, B, C, a, b, c\}, \{a, b, c\}, P, S) \\
P &= \{S \to AB, A \to aA, A \to bA, B \to aB, B \to bB, A \to a, A \to b, B \to a, B \to b\} \\
T &= \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\} \\
t_1 &= (S \to abc, \{\}) \\
t_2 &= (S \to aAbBcC, \{t_3, t_6\}) \\
t_3 &= (A \to aA, \{t_4\}) \\
t_4 &= (B \to bB, \{t_5\}) \\
t_5 &= (B \to cC, \{t_3, t_6\}) \\
t_6 &= (A \to a, \{t_7\}) \\
t_7 &= (B \to b, \{t_8\}) \\
t_8 &= (C \to c, \{\}) \\
L(G_5) &= \{a^n b^n c^n \mid n \geq 1\}
\end{aligned}$$

A derivação abaixo é tal que $p_{i_{j+1}} \in s_{i_j}, j \geq 1$:

$$\begin{aligned}
&S \Rightarrow_{p_2} aAbBcC \Rightarrow_{p_3} aaAbBcC \Rightarrow_{p_4} aaAbbBcC \Rightarrow_{p_5} \\
&aaAbbBccC \Rightarrow_{p_6} aaabbBccC \Rightarrow_{p_7} aaabbbccC \Rightarrow_{p_8} aaabbbccc
\end{aligned}$$

De fato, $t_3 \in s_2, t_4 \in s_3, t_5 \in s_4, t_6 \in s_5, t_7 \in s_6$ e $t_8 \in s_7$. □

Exemplo 5.11

$$\begin{aligned}
G_6 &= (G'_6, T\}) \\
G'_6 &= (\{S, A, B, a, b\}, \{a, b\}, P, S) \\
P &= \{S \to AB, A \to aA, A \to bA, B \to aB, B \to bB, A \to a, A \to b, B \to a, B \to b\} \\
T &= \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9\} \\
t_1 &= (S \to AB, \{t_2, t_3, t_6, t_7\}) \\
t_2 &= (A \to aA, \{t_4\}) \\
t_3 &= (A \to bA, \{t_5\}) \\
t_4 &= (B \to aB, \{t_2, t_3, t_6, t_7\}) \\
t_5 &= (B \to bB, \{t_2, t_3, t_6, t_7\}) \\
t_6 &= (A \to a, \{t_8\}) \\
t_7 &= (A \to b, \{t_9\})
\end{aligned}$$

$$
\begin{aligned}
t_8 &= (B \rightarrow a, \{\}) \\
t_9 &= (B \rightarrow b, \{\}) \\
L(G_6) &= \{ww \mid w \in \{a,b\}^+\}
\end{aligned}
$$

A derivação abaixo é tal que $p_{i_{j+1}} \in s_{i_j}, j \geq 1$:

$$
\begin{aligned}
&S \Rightarrow_{p_1} AB \Rightarrow_{p_2} aAB \Rightarrow_{p_4} aAaB \Rightarrow_{p_3} \\
&abAaB \Rightarrow_{p_5} abAabB \Rightarrow_{p_7} abbabB \Rightarrow_{p_9} abbabb
\end{aligned}
$$

De fato, $t_2 \in s_1, t_4 \in s_2, t_3 \in s_4, t_5 \in s_3, t_7 \in s_5$ e $t_9 \in s_7$. □

Gramáticas periodicamente variantes no tempo

Correspondem a gramáticas $G = (V, \Sigma, P, S)$ que são complementadas por uma estrutura U, composta por subconjuntos $P_i \subseteq P, 1 \leq i \leq k$. Ao se associar cada passo de derivação a um número inteiro, iniciando em 1, a estrutura U determina quais regras de produção podem ser usadas em tal etapa de derivação. A periodicidade da variação no tempo é definida pelo valor de k, de tal forma que no passo de derivação i são consideradas apenas as regras de produção pertencentes ao subconjunto $P_{(i \ mod \ k)}$. Gramáticas variantes no tempo foram propostas pela primeira vez em [45]. Uma gramática livre de contexto periodicamente variante no tempo G é definida formalmente como:

$$
\begin{aligned}
G &= (G', U) \\
G' &= (V, \Sigma, P, S) \text{ é uma gramática livre de contexto} \\
U &= \{P_1, P_2, ..., P_k\}, k \geq 1, P_i \subseteq P, 1 \leq i \leq k \\
L(G) &= \{w \in \Sigma^* \mid \\
&\quad (S \Rightarrow_{p_{i_1}} w_1 \Rightarrow_{p_{i_2}} ... \Rightarrow_{p_{i_q}} w) \text{ e} \\
&\quad (p_{i_j} \in P_{(j \ mod \ k)}, 1 \leq j \leq q)\}
\end{aligned}
$$

Exemplo 5.12

$$
\begin{aligned}
G_7 &= (G'_7, U) \\
G'_7 &= (\{S, A, B, C, X, a, b, c\}, \{a, b, c\}, P, S) \\
P &= \{S \rightarrow ABC, C \rightarrow cC, C \rightarrow c, \\
&\quad A \rightarrow aA, A \rightarrow a, X \rightarrow b, \\
&\quad B \rightarrow bB, B \rightarrow X\} \\
U &= \{P_1, P_2, P_3\} \\
P_1 &= \{S \rightarrow ABC, C \rightarrow cC, C \rightarrow c\} \\
P_2 &= \{A \rightarrow aA, A \rightarrow a, X \rightarrow b\} \\
P_3 &= \{B \rightarrow bB, B \rightarrow X\} \\
L(G_7) &= \{a^n b^n c^n \mid n \geq 1\}
\end{aligned}
$$

A derivação abaixo é tal que $p_{i_j} \in P_{(j \ mod \ 3)}$:

$$
\begin{aligned}
&S \Rightarrow ABC \Rightarrow aABC \Rightarrow aAbBC \Rightarrow \\
&aAbBcC \Rightarrow aabBcC \Rightarrow aaabXcC \Rightarrow aabXcc \Rightarrow aabbcc
\end{aligned}
$$

De fato, as regras de produção aplicadas nessa seqüência de derivações pertencem, respectivamente, aos conjuntos $P_1, P_2, P_3, P_1, P_2, P_3, P_1$ e P_2. □

Exemplo 5.13

$$
\begin{aligned}
G_8 &= (G'_8, U) \\
G'_8 &= (\{S, X_1, X_2, Y_1, Y_2, a, b\}, \{a, b\}, P, S) \\
P &= \{S \to X_1\,Y_1, X_1 \to X_1, Y_2 \to Y_2, \\
&\quad X_1 \to aX_1, X_1 \to bX_2, X_1 \to \epsilon, X_2 \to aX_1, X_2 \to bX_2, X_2 \to \epsilon, \\
&\quad Y_1 \to a\,Y_1, Y_1 \to b\,Y_2, Y_1 \to \epsilon, Y_2 \to a\,Y_1, Y_2 \to b\,Y_2, Y_2 \to \epsilon, \\
&\quad X_2 \to X_2, Y_1 \to Y_1\} \\
U &= \{P_1, P_2, P_3, P_4\} \\
P_1 &= \{S \to X_1\,Y_1, X_1 \to X_1, Y_2 \to Y_2\} \\
P_2 &= \{X_1 \to aX_1, X_1 \to bX_2, X_1 \to \epsilon, X_2 \to aX_1, X_2 \to bX_2, X_2 \to \epsilon\} \\
P_3 &= \{Y_1 \to a\,Y_1, Y_1 \to b\,Y_2, Y_1 \to \epsilon, Y_2 \to a\,Y_1, Y_2 \to b\,Y_2, Y_2 \to \epsilon\} \\
P_4 &= \{X_2 \to X_2, Y_1 \to Y_1\} \\
L(G_8) &= \{ww \mid w \in \{a, b\}^+\}
\end{aligned}
$$

A derivação abaixo é tal que $p_{i_j} \in P_{(j \mod 4)}$:

$$
\begin{aligned}
&S \Rightarrow X_1\,Y_1 \Rightarrow aX_1\,Y_1 \Rightarrow aX_1\,aY_1 \Rightarrow aX_1\,aY_1 \Rightarrow \\
&abX_2\,aY_1 \Rightarrow abX_2\,ab\,Y_2 \Rightarrow abX_2\,ab\,Y_2 \Rightarrow abX_2\,ab\,Y_2 \Rightarrow abbX_2\,ab\,Y_2 \Rightarrow \\
&abbX_2\,abb\,Y_2 \Rightarrow abbX_2\,abb\,Y_2 \Rightarrow abbX_2\,abb\,Y_2 \Rightarrow abbabb\,Y_2 \Rightarrow abbabb
\end{aligned}
$$

De fato, as regras de produção aplicadas nessa seqüência de derivações pertencem, respectivamente, aos conjuntos $P_1, P_2, P_3, P_4, P_1, P_2, P_3, P_4, P_1, P_2, P_3, P_4, P_1, P_2$ e P_3. □

O interesse e a pesquisa sobre gramáticas com mecanismos controladores de derivações não é novo, tendo ocorrido principalmente entre meados da década de 1960 e o início da década de 1970. Apesar de se tratar de uma área pouco considerada nos dias de hoje, tanto do ponto de vista teórico quanto do ponto de vista tecnológico, não se pode negar que a elevada simplicidade dos seus formalismos, aliada à abrangência da classe de linguagens por eles representáveis, as torna dignas de referência e estudo na teoria clássica de linguagens formais. Não obstante, o formalismo gramatical tradicional ainda é mais adequado na demonstração dos resultados teóricos considerados nesse texto, e por esse motivo volta-se a adotá-lo nas seções seguintes do presente texto.

5.3 Formas Normais para Gramáticas Sensíveis ao Contexto

A representação formal das dependências de contexto de uma linguagem pode ser efetuada apenas, conforme discutido, por intermédio das gramáticas do tipo 1 ou, naturalmente, através das de tipo 0.

É conveniente, no entanto, considerar uma importante forma normal para a representação de gramáticas do tipo 1. Nesta forma, as regras são todas reescritas em conformidade com o seguinte padrão:

$$\alpha A \gamma \to \alpha \beta \gamma$$

com $A \in N, \beta \in V^+$ e $\alpha, \gamma \in V^*$.

Demonstra-se a seguir (Teorema 5.1) que toda e qualquer gramática do tipo 1 pode ser convertida para uma nova gramática em que todas as produções obedecem ao formato

acima apresentado, exceto, naturalmente, a produção $S \rightarrow \epsilon$, caso a cadeia vazia faça parte da linguagem.

Isso feito, pode-se reinterpretar as produções como especificações de substituições para um determinado não-terminal A por β, apenas quando A estiver cercado das cadeias α e γ, ou seja, quando A estiver no **contexto** de α e γ.

Portanto, diz-se que a substituição de A por β **depende** da ocorrência de um contexto esquerdo α e de um contexto direito γ para o não-terminal A, fato este que motiva o emprego do termo alternativo **dependente de contexto** para designar as gramáticas do tipo 1.

Deve-se, por outro lado, perceber que uma condição deste tipo nunca ocorre com as gramáticas do tipo 2, nas quais qualquer substituição de um símbolo não-terminal ocorre sempre de forma independente do contexto em que tal não-terminal é encontrado, motivando dessa maneira o emprego do nome "livre de contexto" para designar tais gramáticas.

Teorema 5.1 (Forma normal para gramáticas sensíveis ao contexo) *Se L é uma linguagem sensível ao contexto, então $L = L(G)$, sendo G uma gramática em que todas as regras são do tipo $uAv \rightarrow uwv, w \in (\Sigma \cup N)^+, u, v \in (\Sigma \cup N)^*, A \in N$.*

Justificativa Conforme o Algoritmo 5.1.

Algoritmo 5.1 (Forma normal para gramáticas sensíveis ao contexo) *Obtenção de forma normal para gramáticas sensíveis ao contexto, em que as regras possuem todas o formato $uAv \rightarrow uwv$.*

- Entrada: uma gramática que representa uma linguagem sensível ao contexto L;
- Saída: uma gramática sensível ao contexto G, tal que todas as suas regras satisfazem ao padrão $uAv \rightarrow uwv, w \in (\Sigma \cup N)^+, u, v \in (\Sigma \cup N)^*, A \in N$ e, além disso, $L = L(G)$;
- Método:

  Considerem-se as regras de G numeradas de 1 a p. Cada regra tem o formato geral:

  $$X_{i1}X_{i2}X_{i3}...X_{im} \rightarrow Y_{i1}\,Y_{i2}\,Y_{i3}...\,Y_{in}$$

  onde $1 \leq i \leq p$ é o índice da regra considerada e $X_{ij}, Y_{ik} \in (\Sigma \cup N)$. Como se trata de uma linguagem sensível ao contexto, L é gerada por uma gramática com regras monotônicas, ou seja, aquelas em que o comprimento do lado direito das regras nunca é menor do que o comprimento do respectivo lado esquerdo. No caso acima, $n \geq m$.

  1. O primeiro passo da transformação envolve a substituição de todos os símbolos terminais a_k da gramática por novos símbolos não-terminais A_k correspondentes, acrescentando-se à gramática a regra $A_k \rightarrow a_k$ para cada terminal assim substituído. Naturalmente, esse procedimento não invalida as considerações do parágrafo anterior.

  2. A seguir, substitui-se cada uma das regras da gramática original por um conjunto equivalente de regras, também monotônicas, porém com a restrição

de que o lado direito de cada nova regra possua comprimento máximo 2. Assim,

Para i variando de 1 até p:

a) Considere-se a regra i no formato:

$$X_{i1}X_{i2}X_{i3}...X_{im} \rightarrow Y_{i1}Y_{i2}Y_{i3}...Y_{in}$$

b) Prossiga-se com a regra i caso ($m \geq 3$) ou ($m = 2$ e $n \geq 3$); caso contrário, desvie-se para o início do laço;

c) Crie-se um conjunto de $n-1$ símbolos não-terminais denotados por Z_{i1} até $Z_{i(n-1)}$;

d) Substitua-se a regra original pelo seguinte conjunto de regras:

$$\begin{aligned}
X_{i1}X_{i2} &\rightarrow Y_{i1}Z_{i1} \\
Z_{i1}X_{i3} &\rightarrow Y_{i2}Z_{i2} \\
&\cdots \\
Z_{i(m-2)}X_{im} &\rightarrow Y_{i(m-1)}Z_{i(m-1)} \\
Z_{i(m-1)} &\rightarrow Y_{im}Z_{im} \\
Z_{im} &\rightarrow Y_{i(m+1)}Z_{i(m+1)} \\
Z_{i(m+1)} &\rightarrow Y_{i(m+2)}Z_{i(m+2)} \\
&\cdots \\
Z_{i(n-2)} &\rightarrow Y_{i(n-1)}Z_{i(n-1)} \\
Z_{i(n-1)} &\rightarrow Y_{in}
\end{aligned}$$

Ao término desta etapa, todas as regras da gramática seguirão um dos seguintes formatos:

(i) $PQ \rightarrow RS$

(ii) $P \rightarrow R^{+}$

com $P, Q \in N$ e $R, S \in (\Sigma \cup N)$.

Note-se que o lado esquerdo dessas regras contém apenas símbolos não-terminais. As regras do tipo (ii) já estarão no formato $uAv \rightarrow uwv$ (basta considerar $u, v = \epsilon$), e portanto não há mais nada a fazer com elas. As regras do tipo (i), no entanto, ainda precisam ser convertidas para o formato desejado.

Considerem-se agora as regras numeradas de 1 a q. Considerem-se, dentre estas, as regras que obedecem ao formato (i) citado acima:

$$P_{j1}Q_{j2} \rightarrow R_{j1}S_{j2}$$

onde $1 \leq j \leq q$ é o índice da regra considerada, $P_{j1}, Q_{j2} \in N, R_{j1}, S_{j2} \in (\Sigma \cup N)$. Pode-se aplicar a seguinte transformação de modo a substituir a regra original por um conjunto de regras equivalentes que obedecem ao formato $uAv \rightarrow uwv$:

3. Para j variando de 1 até q:

   a) Considere-se a regra j;

   b) Prossiga-se com a regra j caso ela esteja no formato $P_{j1}\,Q_{j2} \rightarrow R_{j1}\,S_{j2}$; caso contrário, retorne-se ao início do laço;

   c) Crie-se um novo símbolo não-terminal, W_j;

   d) Substitua-se a regra original pelo seguinte conjunto de regras:

$$\begin{aligned} P_{j1}\,Q_{j2} &\rightarrow P_{j1}\,W_j \\ P_{j1}\,W_j &\rightarrow R_{j1}\,W_j \\ R_{j1}\,W_j &\rightarrow R_{j1}\,S_{j2} \end{aligned}$$

Demonstra-se que a gramática resultante pela aplicação do Algoritmo 5.1 gera a mesma linguagem L e todas as suas regras obedecem ao formato $uAv \rightarrow uwv$. ■

Exemplo 5.14
A gramática abaixo representa a linguagem $\{a^n\,b^n\,c^n \mid n \geq 1\}$.

$$\begin{aligned} S &\rightarrow abc && (5.1)\\ S &\rightarrow aSQ && (5.2)\\ bQc &\rightarrow bbcc && (5.3)\\ cQ &\rightarrow Qc && (5.4) \end{aligned}$$

1. *Primeiro passo*: substituição dos símbolos terminais. O seguinte novo conjunto de regras é obtido:

$$\begin{aligned} S &\rightarrow ABC && (5.5)\\ A &\rightarrow a && (5.6)\\ B &\rightarrow b && (5.7)\\ C &\rightarrow c && (5.8)\\ S &\rightarrow ASQ && (5.9)\\ BQC &\rightarrow BBCC && (5.10)\\ CQ &\rightarrow QC && (5.11) \end{aligned}$$

2. *Segundo passo*: Todas essas regras satisfazem ao critério de seleção, exceto a Regra (5.10). Procede-se, então, à sua manipulação.

- Regra (5.10): criam-se os novos símbolos não-terminais Z_{61}, Z_{62} e Z_{63}, substituindo a regra original pelo conjunto de regras:

$$\begin{aligned} BQ &\rightarrow BZ_{61} && (5.12)\\ Z_{61}C &\rightarrow BZ_{62} && (5.13)\\ Z_{62} &\rightarrow CZ_{63} && (5.14)\\ Z_{63} &\rightarrow C && (5.15)\\ & && (5.16) \end{aligned}$$

Chega-se, portanto, à seguinte gramática modificada:

$$\begin{aligned} S &\rightarrow ABC && (5.17)\\ A &\rightarrow a && (5.18)\\ B &\rightarrow b && (5.19)\\ C &\rightarrow c && (5.20)\\ S &\rightarrow ASQ && (5.21)\\ BQ &\rightarrow BZ_{61} && (5.22)\\ Z_{61}C &\rightarrow BZ_{62} && (5.23)\\ Z_{62} &\rightarrow CZ_{63} && (5.24)\\ Z_{63} &\rightarrow C && (5.25)\\ CQ &\rightarrow QC && (5.26) \end{aligned}$$

3. *Terceiro passo*: Com exceção das Regras (5.23) e (5.26), todas as demais satisfazem ao critério de seleção.

   - Regra (5.23): $Z_{61}C \rightarrow BZ_{62}$ é substituída por:

$$\begin{aligned} Z_{61}C &\rightarrow Z_{61}W_7 && (5.27)\\ Z_{61}W_7 &\rightarrow BW_7 && (5.28)\\ BW_7 &\rightarrow BZ_{62} && (5.29) \end{aligned}$$

   - Regra (5.26): $CQ \rightarrow QC$ é substituída por:

$$\begin{aligned} CQ &\rightarrow CW_{10} && (5.30)\\ CW_{10} &\rightarrow QW_{10} && (5.31)\\ QW_{10} &\rightarrow QC && (5.32) \end{aligned}$$

A versão final da gramática desejada torna-se:

$$\begin{aligned} S &\rightarrow ABC && (5.33)\\ A &\rightarrow a && (5.34)\\ B &\rightarrow b && (5.35)\\ C &\rightarrow c && (5.36)\\ S &\rightarrow ASQ && (5.37)\\ BQ &\rightarrow BZ_{61} && (5.38)\\ Z_{61}C &\rightarrow Z_{61}W_7 && (5.39) \end{aligned}$$

$$Z_{61}W_7 \rightarrow BW_7 \quad (5.40)$$
$$BW_7 \rightarrow BZ_{62} \quad (5.41)$$
$$Z_{62} \rightarrow CZ_{63} \quad (5.42)$$
$$Z_{63} \rightarrow C \quad (5.43)$$
$$CQ \rightarrow CW_{10} \quad (5.44)$$
$$CW_{10} \rightarrow QW_{10} \quad (5.45)$$
$$QW_{10} \rightarrow QC \quad (5.46)$$

□

Uma outra importante forma normal para as gramáticas sensíveis ao contexto é a Forma Normal de Kuroda. Apesar de não evidenciar diretamente os contextos em que são feitas as substituições, como no caso da forma normal anterior, ela é utilizada em certas demonstrações teóricas. Formalmente, diz-se que uma gramática sensível ao contexto encontra-se na **Forma Normal de Kuroda** se todas as suas produções $\alpha \rightarrow \beta$ estiverem em conformidade com alguma das seguintes condições:

- $\alpha \in N$ e $\beta \in \Sigma$;
- $\alpha \in N$ e $\beta \in N$;
- $\alpha \in N$ e $\beta \in NN$;
- $\alpha \in NN$ e $\beta \in NN$;

O Teorema 5.2 estabelece a possibilidade de obtenção da Forma Normal de Kuroda para gramáticas sensíveis ao contexto. Naturalmente, ela se aplica apenas a linguagens sensíveis ao contexto que não contenham a cadeia vazia.

Teorema 5.2 (Forma Normal de Kuroda) *Obtenção de uma gramática na Forma Normal de Kuroda equivalente a uma gramática sensível ao contexto G qualquer, desde que* $\epsilon \notin L(G)$.

Justificativa Pode ser encontrada em [55]. ■

5.4 Máquinas de Turing com Fita Limitada

As Máquinas de Turing com fita limitada — também conhecidas como "Autômatos com Limitação Linear" (da fita de trabalho), ou como *"Linear Bounded Automata"*, em inglês — recebem esta denominação alternativa pelo fato de ser possível demonstrar (ver [46]) que linguagens sensíveis ao contexto podem ser reconhecidas por autômatos do tipo Máquina de Turing em que o tamanho da fita de trabalho é uma função linear do tamanho da cadeia a ser analisada. Se o comprimento da cadeia de entrada for n, existe algum k inteiro, maior ou igual a 1, tal que com uma fita de trabalho limitada a um máximo de $k * n$ posições, incluindo a cadeia de entrada, seja possível efetuar a sua análise.

Mais do que isso, pode-se demonstrar (também em [46]) que, com apenas $n + 2$ posições na fita de trabalho, também é possível obter os mesmos resultados que se teria com uma fita limitada proporcionalmente ao tamanho da cadeia de entrada. As n posições intermediárias da fita sendo utilizadas para armazenar a cadeia de entrada e as duas adicionais referindo-se aos delimitadores da referida cadeia, um mais à esquerda e outro mais à direita da mesma.

Uma **Máquina de Turing com fita limitada** é um dispositivo não-determinístico de reconhecimento de cadeias que possui algumas importantes extensões em relação aos autômatos finitos ou aos autômatos de pilha. As mais importantes são:

1. A fita de trabalho possui comprimento igual ao comprimento da cadeia de entrada, acrescido de dois (uma posição para indicar o início da cadeia e outra para indicar seu término; tais indicações são feitas através de símbolos especiais, não pertencentes ao alfabeto de entrada);

2. O cursor de acesso aos símbolos da fita de trabalho pode se deslocar, sob o comando do controle finito, tanto para a direita quanto para a esquerda;

3. Através do cursor de acesso pode-se não apenas ler os símbolos contidos na posição corrente da fita de trabalho, como também gravar novos símbolos em substituição ao símbolos existentes.

Note-se que, diferentemente do que acontece nas máquinas estudadas anteriormente, a fita contendo a cadeia a ser analisada é uma **fita de trabalho**, e não simplesmente de entrada, já que ela pode sofrer alterações durante a operação da Máquina de Turing.

Formalmente, uma Máquina de Turing com fita limitada M é definida como:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, <, >, F)$$

onde:

- Q é o conjunto finito de estados;
- Σ é o alfabeto de entrada, composto por um conjunto finito de símbolos;
- Γ é o conjunto, também finito, de símbolos que podem ser lidos e/ou gravados na fita de trabalho. $\Sigma \subseteq \Gamma$;
- δ é a função parcial de transição, compreendendo os seguintes mapeamentos:
  - $Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{E,D\}}$
  - $Q \times \{<\} \rightarrow 2^{Q \times \{<\} \times \{D\}}$
  - $Q \times \{>\} \rightarrow 2^{Q \times \{>\} \times \{E\}}$
- q_0 é o estado inicial, $q_0 \in Q$;
- $<, > \notin \Gamma$ são símbolos respectivamente situados imediatamente à esquerda e imediatamente à direita da cadeia de entrada na configuração inicial;
- $F \subseteq Q$ é o conjunto de estados finais.

Deve-se, inicialmente, perceber a caracterização de dois alfabetos distintos neste formalismo. Σ representa o conjunto dos símbolos que compõem as cadeias de entrada. A linguagem de entrada deve ser, naturalmente, um subconjunto de Σ^*. Por outro lado, Γ representa o conjunto dos símbolos que podem ser lidos e/ou gravados na fita

de trabalho. Este conjunto incorpora, pela sua própria definição, o conjunto Σ, além de outros símbolos que serão utilizados durante a operação do dispositivo.

O conjunto Q e o estado q_0 representam, respectivamente, assim como no caso dos autômatos finitos e de pilha, o conjunto de estados que formam o controle finito do dispositivo e o estado inicial — único — que é utilizado para caracterizar a configuração inicial do mesmo. F representa um subconjunto de Q e contém os chamados estados finais, que são utilizados para caracterizar as configurações finais em um dispositivo deste tipo.

Os símbolos "<" e ">" são empregados como **delimitadores** da cadeia a ser analisada na fita de trabalho. Eles não pertencem ao alfabeto Γ, indicando com isso que não podem ser utilizados na composição das cadeias de entrada nem ser gravados na fita de trabalho em sobreposição a outros símbolos (exceto em sobreposição a si próprios).

A inspeção da função de transição δ revela que os símbolos especiais $<$ e $>$ só podem ser gravados na própria posição original em que se encontravam no início da operação da máquina. A função δ, da maneira como é formulada, impede ainda que outro símbolo seja gravado em qualquer uma dessas duas posições. Ela especifica, ainda, um único sentido de movimentação do cursor em cada caso: esquerda para ">" e direita para "<".

A representação da função de transição δ indica, a partir do estado corrente e do símbolo correntemente selecionado na fita de trabalho, o novo estado corrente, o novo símbolo que deverá substituir o correntemente selecionado e o sentido em que o cursor de acesso deverá se deslocar: para a esquerda (E) ou para a direita (D).

Considere-se a transição $\delta(q_i, \sigma_m) = \{(q_j, \sigma_n, E)\}$. As seguintes ações são tomadas, nesta seqüência, após a seleção dessa transição:

- O estado corrente q_i é substituído pelo novo estado q_j;
- O símbolo correntemente apontado pelo cursor de acesso, σ_m, é substituído, na fita de trabalho, pelo novo símbolo σ_n;
- O cursor de acesso é deslocado de uma posição para a esquerda (E).

Como regra adicional, deve-se registrar que as Máquinas de Turing com fita limitada não permitem movimentos do cursor de acesso à esquerda da posição preenchida com "<", nem à direita da posição preenchida com ">". Sua movimentação fica, portanto, confinada ao trecho da fita de trabalho situado entre os delimitadores "<" e ">", o que justifica o uso do termo "fita limitada" para caracterizar esse tipo de dispositivo.

A definição da função δ remete ainda para o não-determinismo inerente a esse tipo de dispositivo. Dada uma mesma combinação de estado corrente e de símbolo na fita de trabalho, é possível especificar múltiplas transições envolvendo o próximo estado, o novo símbolo a ser gravado na fita e o sentido de movimentação do cursor de acesso após a gravação.

Apesar de ser possível se considerar a definição de Máquinas de Turing com fita limitada determinísticas, a questão de uma eventual equivalência entre as Máquinas de Turing com fita limitada não-determinísticas e uma correspondente versão determinística é um problema em aberto, de solução ainda desconhecida ([58]). Vale lembrar que este resultado foi provado verdadeiro para o caso dos autômatos finitos e falso para o caso dos autômatos de pilha.

A **configuração** de uma Máquina de Turing com fita limitada é denotada através da tripla (α, q_k, β), em que q_k é o estado corrente, $\alpha \in \{<\}\Gamma^*$ é a porção da cadeia de entrada que se encontra à esquerda do cursor de acesso e $\beta \in \Gamma^*\{>\}$ é a porção da cadeia

de entrada que se encontra à direita do cursor de acesso, incluindo a posição por ele correntemente selecionada. Note-se que "<" e ">" podem ocorrer, cada um, no máximo uma vez em $\alpha\beta$, e sempre nos respectivos extremos.

A **configuração inicial** é $(<, q_0, \gamma >)$, onde q_0 é o estado inicial e $\gamma \in \Sigma^*$ é a cadeia de entrada a ser analisada. O cursor de acesso refere-se, portanto, ao símbolo inicial (mais à esquerda) da cadeia γ. A porção α da representação (α, q_k, β) corresponde, neste caso, apenas ao símbolo "<", pois não existe fita à esquerda deste delimitador. A configuração final é definida como (λ, q_f, μ), com $q_f \in F, \lambda \in \{<\}\Gamma^*$ e $\mu \in \Gamma^*\{>\}$.

As transições contidas na função δ especificam possibilidades de movimentação, que conduzem o dispositivo de cada possível configuração para a correspondente configuração seguinte. Diz-se que o dispositivo **pára** quando a função δ não estiver definida para o par (estado, símbolo de entrada) corrente.

A linguagem aceita por uma Máquina de Turing com fita limitada é o conjunto de todas as cadeias que são capazes de conduzir o dispositivo desde a sua configuração inicial (única para cada cadeia de entrada) até uma configuração final qualquer, sem possibilidade de movimentação adicional. Formalmente:

$$L(M) = \{\gamma \in \Sigma^* \mid (<, q_0, \gamma >) \vdash^* (\lambda, q_f, \mu), \text{ com } q_f \in F, \lambda \in \{<\}\Gamma^* \text{ e } \mu \in \Gamma^*\{>\}\}$$

Admite-se, como condição de parada, que $\mu = \sigma\pi$, com $\sigma \in (\Gamma \cup \{<, >\}), \pi \in (\Gamma^*\{>\} \cup \{\epsilon\})$ e δ não seja definida para (q_f, σ).

Deve-se, por último, notar que, diferentemente dos autômatos finitos e dos autômatos de pilha, as Máquinas de Turing com fita limitada não exigem, como pré-requisito para a caracterização de uma configuração final, que a cadeia de entrada tenha sido esgotada ou, ainda, que o cursor de acesso se encontre à direita do último símbolo da referida cadeia. Configurações finais são caracterizadas quando (i) não há transição possível de ser aplicada na configuração corrente e (ii) o estado corrente é final, não importando a posição do cursor de acesso.

Exemplo 5.15 A Máquina de Turing com fita limitada $M = (Q, \Sigma, \Gamma, \delta, q_0, <, >, F)$ mostrada na Figura 5.1 aceita a linguagem a^*b^*.

$$\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{a, b\} \\
\delta &= \{(q_0, a) \to (q_0, a, D), (q_0, b) \to (q_1, b, D), (q_0, >) \to (q_2, >, E), \\
&\quad (q_1, b) \to (q_1, b, D), (q_1, >) \to (q_1, >, E)\} \\
F &= \{q_2\}
\end{aligned}$$

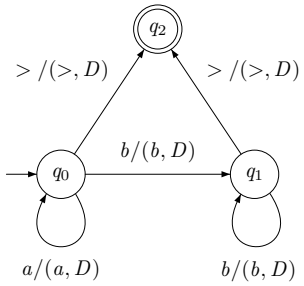

Figura 5.1: Máquina de Turing com fita limitada que aceita a^*b^*

- Exemplo de cadeia reconhecida: $aabbb$

  $(<, q_0, aabbb\ >) \vdash (<\ a, q_0, abbb\ >) \vdash (<\ aa, q_0, bbb\ >) \vdash (<\ aab, q_1, bb\ >) \vdash (<\ aabb, q_1, b >) \vdash (<\ aabbb, q_1, >) \vdash (<\ aabb, q_2, b >)$

  Como não há movimentação possível a partir da configuração $(<\ aabb, q_2, b >)$, que é final, a máquina pára e a cadeia $aabbb$ é aceita.

- Exemplo de cadeia rejeitada: $aaba$

  $(<, q_0, aaba >) \vdash (< a, q_0, aba >) \vdash (< aa, q_0, ba >) \vdash (< aab, q_1, a >)$

  Como não há movimentação possível a partir da configuração $(<\ aab, q_1, a >)$, que não é final, a máquina pára e a cadeia $aaba$ é rejeitada.

A Máquina de Turing com fita limitada do Exemplo 5.15 comporta-se como um autômato finito: o cursor de acesso move-se em apenas um sentido (exceto quando o símbolo "$>$" é encontrado na fita de trabalho) e não há gravação de símbolos na fita, apenas leitura (na verdade, cada símbolo lido é substituído por ele mesmo). Este exemplo sugere que as Máquinas de Turing com fita limitada possam ser usadas em substituição aos autômatos finitos, ainda que com um custo maior de processamento (o custo de gravação de um símbolo na fita).

É fácil, também, perceber como as Máquinas de Turing com fita limitada podem ser empregadas em mecanismos de transdução, à maneira dos transdutores finitos estudados anteriormente.

Se, neste exemplo, as transições:

$$\begin{aligned}(q_0, a) &\rightarrow (q_0, a, D)\\ (q_0, b) &\rightarrow (q_1, b, D)\\ (q_1, b) &\rightarrow (q_1, b, D)\end{aligned}$$

forem, respectivamente, substituídas por:

$$\begin{aligned}(q_0, a) &\rightarrow (q_0, b, D)\\ (q_0, b) &\rightarrow (q_1, a, D)\\ (q_1, b) &\rightarrow (q_1, a, D)\end{aligned}$$

então o autômato resultante mapeará elementos do conjunto a^*b^* em elementos do conjunto b^*a^*, em que os símbolos "a" das cadeias do primeiro conjunto são substituídos por símbolos "b" nas cadeias do segundo conjunto e vice-versa. □

Exemplo 5.16 A Máquina de Turing com fita limitada $M = (Q, \Sigma, \Gamma, \delta, q_0, <, >, F)$ da Figura 5.2 aceita a linguagem $\{a^n b^n \mid n \geq 1\}$.

$$\begin{aligned}Q &= \{q_0, q_1, q_2, q_3, q_4, q_5\}\\ \Sigma &= \{a, b\}\end{aligned}$$

$$
\begin{array}{rcl}
\Gamma & = & \{a, b, X, Y\} \\
\delta & = & \{(q_0, a) \rightarrow (q_1, X, D), (q_0, b) \rightarrow (q_5, b, D), (q_0, Y) \rightarrow (q_3, Y, D), \\
 & & (q_1, a) \rightarrow (q_1, a, D), (q_1, Y) \rightarrow (q_1, Y, D), (q_1, b) \rightarrow (q_2, Y, E), \\
 & & (q_1, >) \rightarrow (q_5, >, E), (q_2, X) \rightarrow (q_0, X, D), (q_2, Y) \rightarrow (q_2, Y, E), \\
 & & (q_2, a) \rightarrow (q_2, a, E), (q_3, Y) \rightarrow (q_3, Y, D), (q_3, b) \rightarrow (q_5, b, D), \\
 & & (q_3, >) \rightarrow (q_4, >, E)\} \\
F & = & \{q_4\}
\end{array}
$$

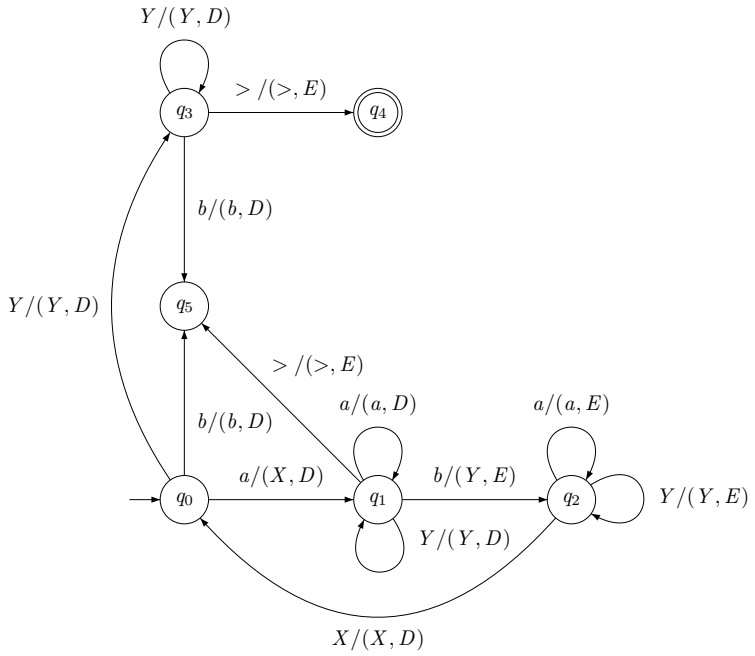

Figura 5.2: Máquina de Turing com fita limitada que aceita $\{a^n b^n \mid n \geq 1\}$

Seu funcionamento é intuitivo e reflete a aplicação do seguinte algoritmo:

1. O símbolo "a" sob o cursor é substituído pelo símbolo "X". O cursor é deslocado de uma posição, para a direita.

2. O cursor continua se deslocando para a direta até encontrar um símbolo "b" ou o símbolo "$>$".

3. Se encontrar "$>$", a cadeia é rejeitada, pois existem mais símbolos "a" do que "b". Se encontrar "b", este será substituído por "Y" e o cursor será deslocado para a esquerda até encontrar o "X" mais à direita. Neste momento, o cursor é deslocado de uma posição para a direita e reinicia-se todo o processo no passo (1).

4. Se o símbolo corrente for "Y", isso indica que já foram considerados todos os símbolos "a". Se o restante da cadeia de entrada for formada apenas por símbolos "Y", ela será aceita. Caso contrário, será rejeitada. É o caso, por exemplo, de cadeias que contêm mais símbolos "b" do que símbolos "a".

A seguir, confere-se a operação de M com algumas cadeias:

- A cadeia $aabb$ é aceita:

  $(<, q_0, aabb >) \vdash (< X, q_1, abb >) \vdash (< Xa, q_1, bb >) \vdash (< X, q_2, aYb >) \vdash (<, q_2, XaYb >) \vdash (< X, q_0, aYb >) \vdash (< XX, q_1, Yb >) \vdash (< XXY, q_1, b >) \vdash (< XX, q_2, YY >) \vdash (< X, q_2, XYY >) \vdash (< XX, q_0, YY >) \vdash (< XXY, q_3, Y >) \vdash (< XXYY, q_3, >) \vdash (< XXY, q_4, Y >)$

- A cadeia aab é rejeitada:

  $(<, q_0, aab >) \vdash (< X, q_1, ab >) \vdash (< Xa, q_1, b >) \vdash (< X, q_2, aY >) \vdash (<, q_2, XaY >) \vdash (< X, q_0, aY >) \vdash (< XX, q_1, Y >) \vdash (< XXY, q_1, >) \vdash (< XX, q_5, Y >)$

- A cadeia abb é rejeitada:

  $(<, q_0, abb >) \vdash (< X, q_1, bb >) \vdash (<, q_2, XYb >) \vdash (< X, q_0, Yb >) \vdash (< XY, q_3, b >) \vdash (< XYb, q_5, >)$

Neste exemplo, a Máquina de Turing com fita limitada está sendo utilizada para reconhecer uma linguagem livre de contexto, e tal fato sugere que esse tipo de dispositivo possa ser empregado também no reconhecimento desta categoria de linguagens, em substituição aos autômatos de pilha.

Além de necessitar da movimentação do cursor em ambos os sentidos, neste exemplo a substituição (gravação) de um símbolo do alfabeto de entrada por símbolos que não fazem parte deste alfabeto (no caso, "a" por "X" e "b" por "Y") é essencial para o seu correto funcionamento. □

Exemplo 5.17 A Máquina de Turing com fita limitada da Figura 5.3 reconhece a linguagem $\{wcw \mid w \in \{a, b\}^*\}$ sobre o alfabeto $\{a, b, c\}$:

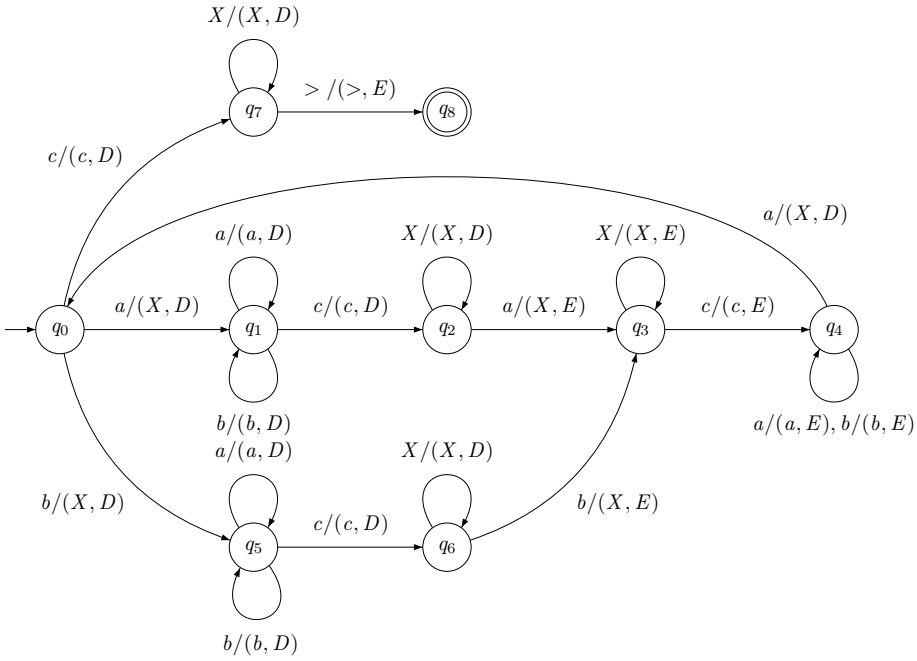

Figura 5.3: Máquina de Turing com fita limitada que aceita $\{wcw \mid w \in \{a, b\}^*\}$

Este exemplo ilustra o reconhecimento de uma linguagem tipicamente sensível ao contexto, pois a subcadeia "w" deve ser a mesma em ambos os lados do símbolo "c". Tal característica remete à relação entre a declaração e o uso de nomes, encontrada na maioria das linguagens de programação algorítmicas tradicionais — um nome só pode ser usado se a sua declaração for visível no local do uso.

- Exemplo de cadeia reconhecida: $abbcabb$

$$
\begin{array}{l}
(<, q_0, abbcabb>) \vdash (< X, q_1, bbcabb >) \vdash (< Xb, q_1, bcabb >) \vdash \\
(< Xbb, q_1, cabb >) \vdash (< Xbbc, q_2, abb >) \vdash (< Xbb, q_3, cXbb >) \vdash \\
(< Xb, q_4, bcXbb >) \vdash (< X, q_4, bbcXbb >) \vdash (<, q_4, XbbcXbb >) \vdash \\
(< X, q_0, bbcXbb >) \vdash (< XX, q_5, bcXbb >) \vdash (< XXb, q_5, cXbb >) \vdash \\
(< XXbc, q_6, Xbb >) \vdash (< XXbcX, q_6, bb >) \vdash (< XXbc, q_3, XXb >) \vdash \\
(< XXb, q_3, cXXb >) \vdash (< XX, q_4, bcXXb >) \vdash (< X, q_4, XbcXXb >) \vdash \\
(< XX, q_0, bcXXb >) \vdash (< XXX, q_5, cXXb >) \vdash (< XXXc, q_6, XXb >) \vdash \\
(< XXXcX, q_6, Xb >) \vdash (< XXXcXX, q_6, b >) \vdash (< XXXcX, q_3, XX >) \vdash \\
(< XXXc, q_3, XXX >) \vdash (< XXX, q_3, cXXX >) \vdash (< XX, q_4, XcXXX >) \vdash \\
(< XXX, q_0, cXXX >) \vdash (< XXXc, q_7, XXX >) \vdash (< XXXcX, q_7, XX >) \vdash \\
(< XXXcXX, q_7, X >) \vdash (< XXXcXXX, q_7, >) \vdash (< XXXcXX, q_8, X >)
\end{array}
$$

Pode-se demonstrar, através do "Pumping Lemma" para linguagens livres de contexto, que a linguagem deste exemplo não é livre de contexto (a demonstração é semelhante à do Exemplo 4.39). Tal resultado sugere, como será mostrado mais adiante, que as Máquinas de Turing com fita limitada são dispositivos capazes de reconhecer uma classe de linguagens mais ampla do que as livres de contexto, reconhecidas pelos autômatos de pilha — trata-se, no caso, da classe das linguagens sensíveis ao contexto, caracterizadas neste capítulo (ver Teorema 5.3). □

5.5 Equivalência entre Gramáticas Sensíveis ao Contexto e Máquinas de Turing com Fita Limitada

Linguagens sensíveis ao contexto podem ser formalizadas através de gramáticas sensíveis ao contexto. As Máquinas de Turing com fita limitada, por outro lado, correspondem aos dispositivos de reconhecimento associados às linguagens sensíveis ao contexto. As gramáticas sensíveis ao contexto e as Máquinas de Turing com fita limitada representam exatamente a mesma classe de linguagens — as linguagens sensíveis ao contexto. Esta equivalência, que foi inicialmente estabelecida por Kuroda em [65], poderá ser melhor percebida através dos algoritmos descritos a seguir.

Teorema 5.3 (Máquina de Turing ⇒ gramática) *Seja $L = L(M), M$ uma Máquina de Turing com fita limitada. Então $L - \{\epsilon\} = L(G)$, com G sendo uma gramática sensível ao contexto.*

Justificativa A idéia geral desta demonstração consiste na obtenção de uma gramática sensível ao contexto que reproduz, na derivação de suas sentenças, os movimentos de uma Máquina de Turing com fita limitada que reconhece a mesma linguagem. Se a cadeia de entrada conduz o autômato a uma configuração final, sendo portanto aceita, esta mesma cadeia é gerada pela gramática. Cadeias rejeitadas pelo autômato não são geradas pela gramática.

O algoritmo descrito a seguir especifica os passos a serem seguidos na transformação sistemática de uma Máquina de Turing com fita limitada $M = (Q, \Sigma, \Gamma, \delta, <, >, F)$ em uma gramática sensível ao contexto $G = (V, \Sigma, P, S)$.

Como se trata de um procedimento canônico, é freqüente que seja gerada uma grande quantidade de símbolos não-terminais e de regras, vários dos quais eventualmente repetidos, mesmo que a Máquina de Turing com fita limitada correspondente possua

poucos estados e poucas entradas. O procedimento descrito abaixo, no entanto, pode ser facilmente automatizado através de programas de computador.

Inicialmente são gerados os símbolos não-terminais de G. Estes símbolos possuem o formato geral $[\sigma, \alpha]$, com $\sigma \in \Sigma$, e $\alpha \in \{<, \epsilon\}\{\sigma, q\sigma, \sigma q\}\{>, \epsilon\}$, com $\sigma \in \Sigma$ e $q \in Q$.

A primeira dessas duas componentes — $[\sigma, ...]$ — representa um certo símbolo terminal da cadeia de entrada. A segunda componente — $[..., \alpha]$ — representa parte da configuração assumida pela Máquina de Turing com fita limitada no reconhecimento da referida cadeia de entrada.

Os conjuntos P e S de G são inicialmente definidos como:

I. $P = \varnothing$

II. $P = P \cup \{S \rightarrow [\sigma_i, < q_0\sigma_i]A, S \rightarrow [\sigma_i, < q_0\sigma_i >], \forall\, \sigma_i \in \Sigma\}$
$N = \{S, A\}$
$N = N \cup \{[\sigma_i, < q_0\sigma_i], \forall\, \sigma_i \in \Sigma\}$
$N = N \cup \{[\sigma_i, < q_0\sigma_i >], \forall\, \sigma_i \in \Sigma\}$

III. $P = P \cup \{A \rightarrow [\sigma_i, \sigma_i]A, A \rightarrow [\sigma_i, \sigma_i >], \forall\, \sigma_i \in \Sigma\}$
$N = N \cup \{[\sigma_i, \sigma_i], \forall\, \sigma_i \in \Sigma\}$
$N = N \cup \{[\sigma_i, \sigma_i >], \forall\, \sigma_i \in \Sigma\}$

Eles permitem gerar formas sentenciais do tipo:

A) $[\sigma_i, < q_0\sigma_i >]$, ou

B) $[\sigma_i, < q_0\sigma_i][\sigma_j, \sigma_j][\sigma_k, \sigma_k]...[\sigma_u, \sigma_u >]$

formadas exclusivamente por símbolos não-terminais de G.

A primeira forma sentencial refere-se ao caso de cadeias de entrada cujo comprimento é igual a 1, ao passo que a segunda se refere a casos em que a cadeia de entrada possui comprimento maior do que 1.

Considere-se o segundo caso. Apenas para efeito didático, pode-se construir duas novas cadeias a partir desta forma sentencial. Tomando-se apenas o primeiro elemento de cada símbolo não-terminal, obtém-se a cadeia:

$$[\sigma_i, ...][\sigma_j, ...][\sigma_k, ...]...[\sigma_u, ...] \text{ ou, simplesmente, } \sigma_i\sigma_j\sigma_k...\sigma_u$$

e tomando-se apenas o segundo elemento de cada símbolo não-terminal:

$$[..., < q_0\sigma_i][..., \sigma_j][..., \sigma_k]...[..., \sigma_u >] \text{ ou, simplesmente, } < q_0\sigma_i\sigma_j\sigma_k...\sigma_u >$$

É fácil perceber que a primeira cadeia ($\sigma_i\sigma_j\sigma_k...\sigma_u$) corresponde à cadeia de entrada que é submetida ao reconhecedor e que será objeto de uma tentativa de síntese através da gramática que se pretende construir.

A segunda cadeia ($< q_0\sigma_i\sigma_j\sigma_k...\sigma_u >$) corresponde à configuração inicial da Máquina de Turing limitada, ou seja, àquela em que o estado inicial é q_0 e o cursor de acesso encontra-se posicionado sobre o símbolo σ_i, portanto com toda a cadeia de entrada à sua direita e apenas o símbolo "<" à sua esquerda. Conforme a notação previamente apresentada, a configuração inicial da máquina será:

$$(<, q_0, \sigma_i\sigma_j\sigma_k...\sigma_u >)$$

Dessa maneira, as formas sentenciais dos tipos A e B correspondem às duas configurações iniciais com as quais o autômato poderá se defrontar e representam, em termos gramaticais, uma versão adequada das mesmas para a simulação dos movimentos do reconhecedor pela gramática. Em outras palavras, partindo-se da raiz S, e pela aplicação das regras dos itens II e III, chega-se a formas sentenciais compostas apenas por símbolos não-terminais de G, conforme ilustrado em A e B.

Um conjunto adicional de regras de P deve ser especificado de forma que se possa garantir que, para toda movimentação possível da Máquina de Turing com fita limitada a partir de uma configuração qualquer, haja a possibilidade de uma derivação correspondente em G.

Genericamente, as regras de M enquadram-se em quatro casos (conforme a definição da função de transição δ):

- Caso i: $\delta(q_i, <) = (q_j, <, D)$
- Caso ii: $\delta(q_i, >) = (q_j, >, E)$
- Caso iii: $\delta(q_i, \sigma_m) = (q_j, \sigma_n, D)$
- Caso iv: $\delta(q_i, \sigma_m) = (q_j, \sigma_n, E)$

Suponha-se, a título de exemplo, que a cadeia de entrada de uma Máquina de Turing com fita limitada qualquer seja "*abca*". A sua configuração inicial é, portanto, $(<, q_0, abca >)$. Nos termos da forma sentencial gerada pelas regras iniciais de G, ela será composta pelos seguintes símbolos não-terminais (quatro no total):

$$[a, < q_0 a][b, b][c, c][a, a >]$$

Suponha-se que a regra $\delta(q_0, a) = (q_1, d, D)$ (caso iii) seja aplicada em M. Neste caso, G deve permitir a derivação:

$$[a, < q_0 a][b, b][c, c][a, a >] \Rightarrow [a, < d][b, q_1 b][c, c][a, a >]$$

Note-se que o movimento de M é reproduzido como uma nova forma sentencial em G, de modo que, através da posição de "q_i" — presente em apenas um símbolo não-terminal de cada vez em cada forma sentencial distinta —, determina-se a posição do cursor de acesso sobre a fita de trabalho.

A indicação de estado "q_i" migra, assim, de um símbolo não-terminal para o seguinte (aquele que está à sua direita), ou, ainda, para o anterior (aquele que está à sua esquerda), caso a cabeça de acesso se desloque para a esquerda.

Caso M aplique uma regra que desloque o cursor de acesso para a esquerda, a nova configuração também é representada como uma nova derivação em G. Por exemplo, suponha-se agora que a regra $\delta(q_1, b) = (q_2, e, E)$ (caso iv) seja aplicada na configuração anterior. Então, do ponto de vista de G, a seguinte nova forma sentencial deveria ser gerada:

$$[a, < d][b, q_1 b][c, c][a, a >] \Rightarrow [a, < q_2 d][b, e][c, c][a, a >]$$

Nesta forma sentencial, se $\delta(q_2, d) = (q_3, d, E)$ (caso iv, novamente), então a nova configuração de M é $(\epsilon, q_3, < abca >)$. Para refletir essa nova condição, G deve possuir regras que permitam a derivação:

$$[a, < q_2 d][b, e][c, c][a, a >] \Rightarrow [a, q_3 < d][b, e][c, c][a, a >]$$

Esta última forma sentencial mostra que M pode, eventualmente, deslocar o cursor de acesso para a esquerda, até fazer com que o símbolo corrente seja "$<$". Em situações como esta, o único movimento permitido é para a direita, sem modificação do símbolo lido (conforme a definição da função de transição). Supondo que $\delta(q_3, <) = (q_4, <, D)$ — na verdade, a única transição possível de ser aplicada neste ponto —, caracteriza-se o caso (i), e a nova forma sentencial obtida é:

$$[a, q_3 < d][b, e][c, c][a, a >] \Rightarrow [a, < q_4 d][b, e][c, c][a, a >]$$

O conjunto de regras de G que são capazes de promover tais derivações é, naturalmente, obtido a partir da especificação da função δ. Deve-se, portanto, considerar os casos individuais. A partir de cada um deles, novas produções são formuladas e, a partir destas, novos símbolos não-teminais são acrescentados à gramática. Observe-se que nos casos (iii) e (iv) as transições são feitas com símbolos do alfabeto Γ, ou seja, $\sigma_m, \sigma_n \in \Gamma$.

- Caso i: $\delta(q_i, <) = (q_j, <, D)$

  A cadeia de entrada pode ser unitária (IV) ou não (V):

  IV. $P = P \cup \{[\sigma_m, q_i < \sigma_m >] \rightarrow [\sigma_m, < q_j \sigma_m >], \forall\, \sigma_m \in \Gamma\}$
  $N = N \cup \{[\sigma_m, q_i < \sigma_m >], \forall\, \sigma_m \in \Gamma\}$
  $N = N \cup \{[\sigma_m, < q_j \sigma_m >], \forall\, \sigma_m \in \Gamma\}$

  V. $P = P \cup \{[\sigma_m, q_i < \sigma_m] \rightarrow [\sigma_m, < q_j \sigma_m], \forall\, \sigma_m \in \Gamma\}$
  $N = N \cup \{[\sigma_m, q_i < \sigma_m], \forall\, \sigma_m \in \Gamma\}$
  $N = N \cup \{[\sigma_m, < q_j \sigma_m], \forall\, \sigma_m \in \Gamma\}$

- Caso ii: $\delta(q_i, >) = (q_j, >, E)$

  A cadeia de entrada pode ser unitária (VI) ou não (VII):

  VI. $P = P \cup \{[\sigma_m, < \sigma_n q_i >] \rightarrow [\sigma_m, < q_j \sigma_n >], \forall\, \sigma_m, \sigma_n \in \Gamma\}$
  $N = N \cup \{[\sigma_m, < \sigma_n q_i >], \forall\, \sigma_m, \sigma_n \in \Gamma\}$
  $N = N \cup \{[\sigma_m, < q_j \sigma_n >], \forall\, \sigma_m, \sigma_n \in \Gamma\}$

  VII. $P = P \cup \{[\sigma_m, \sigma_n q_i >] \rightarrow [\sigma_m, q_j \sigma_n >], \forall\, \sigma_m, \sigma_n \in \Gamma\}$
  $N = N \cup \{[\sigma_m, \sigma_n q_i >], \forall\, \sigma_m, \sigma_n \in \Gamma\}$
  $N = N \cup \{[\sigma_m, q_j \sigma_n >], \forall\, \sigma_m, \sigma_n \in \Gamma\}$

- Caso iii: $\delta(q_i, \sigma_m) = (q_j, \sigma_n, D)$

  O símbolo corrente (σ_m) pode:

  - ser o primeiro da cadeia de entrada:
    - cadeia unitária: VIII;
    - cadeia de comprimento 2: IX;
    - cadeia de comprimento maior que 2: X;

- ser o último da cadeia de entrada:
  - cadeia unitária: coincide com VIII;
  - cadeia não-unitária: IX;
- ser o penúltimo da cadeia de entrada:
  - cadeia de comprimento 2: coincide com IX;
  - cadeia de comprimento maior que 2: XII;
- estar em qualquer posição diferente destas:
  - XIII;

VIII. $P = P \cup \{[\sigma_p, < q_i\sigma_m >] \rightarrow [\sigma_p, < \sigma_n q_j >], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_p, < q_i\sigma_m >], \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_p, < \sigma_n q_j >], \sigma_p \in \Gamma\}$

IX. $P = P \cup \{[\sigma_p, < q_i\sigma_m][\sigma_q, \sigma_r >] \rightarrow [\sigma_p, < \sigma_n][\sigma_q, q_j\sigma_r >], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_q, \sigma_r >], \forall\, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, < \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_q, q_j\sigma_r >], \forall\, \sigma_q, \sigma_r \in \Gamma\}$

X. $P = P \cup \{[\sigma_p, < q_i\sigma_m][\sigma_q, \sigma_r] \rightarrow [\sigma_p, < \sigma_n][\sigma_q, q_j\sigma_r], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_q, \sigma_r], \forall\, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, < \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_q, q_j\sigma_r], \forall\, \sigma_q, \sigma_r \in \Gamma\}$

XI. $P = P \cup \{[\sigma_p, q_i\sigma_m >] \rightarrow [\sigma_p, \sigma_n q_j >], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_i\sigma_m >], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_p, \sigma_n q_j >], \sigma_p \in \Gamma\}$

XII. $P = P \cup \{[\sigma_p, q_i\sigma_m][\sigma_q, \sigma_r >] \rightarrow [\sigma_p, \sigma_n][\sigma_q, q_j\sigma_r >], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_q, \sigma_r >], \forall\, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_q, q_j\sigma_r >], \forall\, \sigma_q, \sigma_r \in \Gamma\}$

XIII. $P = P \cup \{[\sigma_p, q_i\sigma_m][\sigma_q, \sigma_r] \rightarrow [\sigma_p, \sigma_n][\sigma_q, q_j\sigma_r], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_q, \sigma_r], \forall\, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_q, q_j\sigma_r], \forall\, \sigma_q, \sigma_r \in \Gamma\}$

- Caso iv: $\delta(q_i, \sigma_m) = (q_j, \sigma_n, E)$

O símbolo corrente (σ_m) pode:

- ser o primeiro da cadeia de entrada:
  - cadeia unitária: XIV;
  - cadeia não-unitária: XV;
- ser o último da cadeia de entrada:
  - cadeia unitária: coincide com XIV;
  - cadeia de comprimento 2: XVI;
  - cadeia de comprimento maior que 2: XVII;
- ser o segundo da cadeia de entrada:
  - cadeia de comprimento 2: coincide com XVI;
  - cadeia de comprimento maior que 2: XVIII;
- estar em qualquer posição diferente destas:
  - XIX;

XIV. $P = P \cup \{[\sigma_p, < q_i\sigma_m >] \rightarrow [\sigma_p, q_j < \sigma_n >], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_p, < q_i\sigma_m >], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_j < \sigma_n >], \forall\, \sigma_p \in \Gamma\}$

XV. $P = P \cup \{[\sigma_p, < q_i\sigma_m] \rightarrow [\sigma_p, q_j < \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_j < \sigma_n], \forall\, \sigma_p \in \Gamma\}$

XVI. $P = P \cup \{[\sigma_p, < \sigma_q][\sigma_r, q_i\sigma_m >] \rightarrow [\sigma_p, < q_j\sigma_q][\sigma_r, \sigma_n >], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, < \sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_r, q_i\sigma_m >], \forall\, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, < q_j\sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_r, \sigma_n >], \forall\, \sigma_r \in \Gamma\}$

XVII. $P = P \cup \{[\sigma_p, \sigma_q][\sigma_r, q_i\sigma_m >] \rightarrow [\sigma_p, q_j\sigma_q][\sigma_r, \sigma_n >], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, \sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_r, q_i\sigma_m >], \forall\, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_j\sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_r, \sigma_n >], \forall\, \sigma_r \in \Gamma\}$

XVIII. $P = P \cup \{[\sigma_p, < \sigma_q][\sigma_r, q_i\sigma_m] \rightarrow [\sigma_p, < q_j\sigma_q][\sigma_r, \sigma_n], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, < \sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$

$$N = N \cup \{[\sigma_r, q_i\sigma_m], \forall\, \sigma_r \in \Gamma\}$$
$$N = N \cup \{[\sigma_p, < q_j\sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$$
$$N = N \cup \{[\sigma_r, \sigma_n], \forall\, \sigma_r \in \Gamma\}$$

XIX. $P = P \cup \{[\sigma_p, \sigma_q][\sigma_r, q_i\sigma_m] \rightarrow [\sigma_p, q_j\sigma_q][\sigma_r, \sigma_n], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, \sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_r, q_i\sigma_m], \forall\, \sigma_r \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_j\sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_r, \sigma_n], \forall\, \sigma_r \in \Gamma\}$

A operação do autômato cessa quando ele atinge uma configuração para a qual não exista movimentação possível, ou seja, quando $\delta(q_i, \sigma_m) = \varnothing$. Nesta situação, se $q_i \in F$, diz-se que o autômato aceita a cadeia de entrada. Caso contrário, diz-se que ele a rejeita.

Condição semelhante precisa ser alcançada através de G. Suponha-se que, quando o autômato pára de se movimentar, a configuração seja $(\alpha, q_i\sigma_m\beta)$. A forma sentencial equivalente em G será $[...][...]...[\sigma_n, q_i\sigma_m][...]$.

Se, nesta situação, $q_i \in F$ e $\delta(q_i, \sigma_m) = \varnothing$, então as seguintes regras devem ser adicionadas à gramática G:

- Caso v: $q_i \in F$ e $\delta(q_i, <) = \varnothing$

XX. $P = P \cup \{[\sigma_p, q_i < \sigma_q] \rightarrow \sigma_p, \forall\, \sigma_p \in \Sigma, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_i < \sigma_q], \forall\, \sigma_p \in \Sigma, \sigma_q \in \Gamma\}$

XXI. $P = P \cup \{[\sigma_p, q_i < \sigma_q >] \rightarrow \sigma_p, \forall\, \sigma_p \in \Sigma, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_p, q_i < \sigma_q >], \forall\, \sigma_p \in \Sigma, \sigma_q \in \Gamma\}$

- Caso vi: $q_i \in F$ e $\delta(q_i, >) = \varnothing$

XXII. $P = P \cup \{[\sigma_p, \sigma_q q_i >] \rightarrow \sigma_p, \forall\, \sigma_p \in \Sigma, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_p, \sigma_q q_i >], \forall\, \sigma_p \in \Sigma, \sigma_q \in \Gamma\}$

XXIII. $P = P \cup \{[\sigma_p, < \sigma_q q_i >] \rightarrow \sigma_p, \forall\, \sigma_p \in \Sigma, \sigma_q \in \Gamma\}$
$N = N \cup \{[\sigma_p, < \sigma_q q_i >], \forall\, \sigma_p \in \Sigma, \sigma_q \in \Gamma\}$

- Caso vii: $q_i \in F$ e $\delta(q_i, \sigma_m) = \varnothing$

XXIV. $P = P \cup \{[\sigma_p, < q_i\sigma_m] \rightarrow \sigma_p, \forall\, \sigma_p \in \Sigma\}$
$N = N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in \Sigma\}$

XXV. $P = P \cup \{[\sigma_p, q_i\sigma_m] \rightarrow \sigma_p, \forall\, \sigma_p \in \Sigma\}$
$N = N \cup \{[\sigma_p, q_i\sigma_m], \forall\, \sigma_p \in \Sigma\}$

XXVI. $P = P \cup \{[\sigma_p, q_i\sigma_m >] \rightarrow \sigma_p, \forall\, \sigma_p \in \Sigma\}$
$N = N \cup \{[\sigma_p, q_i\sigma_m >], \forall\, \sigma_p \in \Sigma\}$

XXVII. $P = P \cup \{[\sigma_p, < q_i\sigma_m >] \rightarrow \sigma_p, \forall\, \sigma_p \in \Sigma\}$
$N = N \cup \{[\sigma_p, < q_i\sigma_m >], \forall\, \sigma_p \in \Sigma\}$

Essas regras dão início ao processo de conversão da cadeia armazenada no lado esquerdo dos não-terminais — e até então inalterada — em uma sentença a ser efetivamente gerada por G. A forma sentencial inicial será obtida através da derivação $[...][...]...[\sigma_n, q_i\sigma_m][...] \Rightarrow [...][...]...\sigma_n[...]$, se $q_i \in F$ e $\delta(q_i, \sigma_m) = \varnothing$.

Finalmente, devem ser acrescentadas mais algumas produções em P, as quais permitam que os não-terminais remanescentes na forma sentencial sejam todos substituídos por símbolos terminais, conforme o valor armazenado no lado esquerdo, no interior do mesmo:

- Caso viii: Substituição de símbolos não-terminais por símbolos terminais.

XXVIII. $P = P \cup \{[\sigma_m, \alpha]\sigma_n \rightarrow \sigma_m\sigma_n, \forall\, \sigma_n \in \Sigma, [\sigma_m, \alpha] \in V\}$

XXIX. $P = P \cup \{\sigma_n[\sigma_m, \alpha] \rightarrow \sigma_n\sigma_m, \forall\, \sigma_n \in \Sigma, [\sigma_m, \alpha] \in V\}$

A simples inspeção do formato das regras geradas nos itens I a XXIX permite concluir que toda e qualquer gramática gerada por esse método é uma gramática sensível ao contexto. Será omitida a demonstração formal da equivalência das Máquinas de Turing com fita limitada com as gramáticas sensíveis ao contexto, que no entanto pode ser encontrada em [52]. ■

Exemplo 5.18 Considere-se uma Máquina de Turing com fita limitada $M = (Q, \Sigma, \Gamma, \delta, q_0, <, >, F)$, conforme apresentado a seguir, e a cadeia de entrada "ab".

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{a, b\} \\ \Gamma &= \{a, b\} \\ \delta &= \{(q_0, a) \rightarrow (q_0, a, D), (q_0, b) \rightarrow (q_1, b, E)\} \\ F &= \{q_1\} \end{aligned}$$

A Figura 5.4 ilustra os movimentos que levam o reconhecedor desde a sua configuração inicial até uma configuração final. À direita, as respectivas formas sentenciais, que são geradas pela aplicação dos casos enumerados anteriormente. Note-se que, por uma questão de economia de espaço, estão mostradas aqui apenas as regras e os símbolos não-terminais relevantes para a geração desta cadeia. Uma geração exaustiva de todos os símbolos e de todas as regras da gramática G equivalente a M tornaria o exemplo exageradamente extenso, e por isso é deixada como exercício para o leitor.

Reconhecedor		**Gramática**		
Configuração	Transição	Forma sentencial	Regra	Item
-	-	S	S → [a, <q_0a] A	II
-	-	[a, <q_0a] A	A → [b, b>]	III
(<, q_0, ab>)	(q_0, a) → (q_0, a, D)	[a, <q_0a] [b, b>]	[a, <q_0a] [b, b>] → [a, <a] [b, q_0b>]	IX
(<a, q_0, b>)	(q_0, b) → (q_1, b, E)	[a, <a] [b, q_0b>]	[a, <a] [b, q_0b>] → [a, < q_1a] [b, b>]	XVI
(<, q_1, ab>)	-	[a, <q_1a] [b, b>]	[a, <q_1a] → a	XXIV
-	-	a [b, b>]	a [b, b>] → ab	XXIX
-	-	ab	-	-

Figura 5.4: Gramática sensível ao contexto simulando Máquina de Turing com fita limitada para o Exemplo 5.18

□

Para dar seqüência ao assunto, é necessário demonstrar que as Máquinas de Turing com fita limitada, cujas fitas sejam compostas por uma única trilha (o que corresponde ao caso visto até o momento), são equivalentes, quanto ao seu poder de reconhecimento, às Máquinas de Turing com múltiplas trilhas na fita de trabalho. Esse caso está ilustrado na Figura 5.5, em que a fita de trabalho compreende três trilhas distintas:

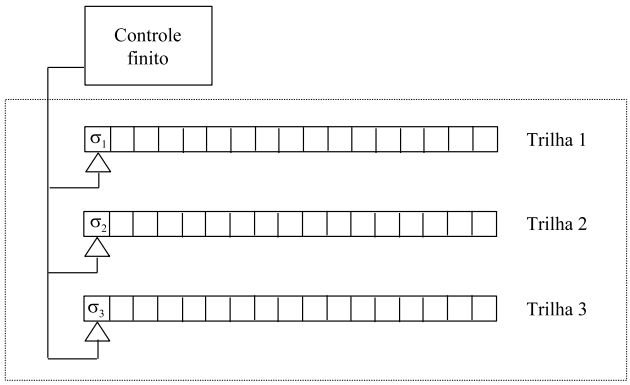

Figura 5.5: Máquina de Turing com múltiplas trilhas na fita de trabalho, situação inicial

Neste modelo, qualquer que seja a quantidade de trilhas na fita de trabalho, os cursores de acesso são sempre deslocados conjuntamente e estão, portanto, sempre na mesma posição da fita, porém cada qual apontando para a célula correspondente. As transições são efetuadas considerando-se o estado corrente e o conjunto dos símbolos referenciados simultaneamente na fita de trabalho.

Na Figura 5.5, tal conjunto corresponde à tripla $(\sigma_1, \sigma_2, \sigma_3)$, em que σ_1 está gravado na primeira posição da trilha 1, σ_2 na primeira posição da trilha 2 e σ_3 na primeira posição da trilha 3. Como resultado da aplicação de uma transição, o autômato muda de estado, a tripla $(\sigma_1, \sigma_2, \sigma_3)$ é substituída por (τ_1, τ_2, τ_3) e os três cursores de acesso deslocam-se simultaneamente de uma posição, por exemplo, à direita (todos juntos). A nova situação pode ser visualizada na Figura 5.6.

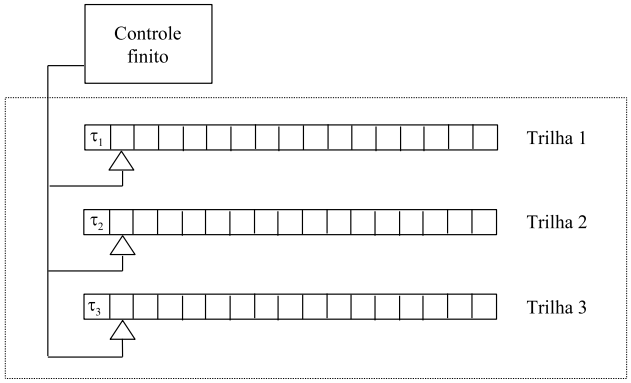

Figura 5.6: Máquina de Turing com múltiplas trilhas na fita de trabalho, situação final

Seja Σ_1 o alfabeto usado na trilha 1, Σ_2 o alfabeto da trilha 2 e Σ_3 o alfabeto da trilha 3. É fácil perceber que a função de transição de um autômato assim caracterizado pode ser expressa na forma:

$$\delta : Q \times (\Sigma_1 \times \Sigma_2 \times \Sigma_3) \rightarrow 2^{Q \times (\Sigma_1 \times \Sigma_2 \times \Sigma_3) \times \{E,D\}}$$

ou, generalizando para n trilhas:

$$\delta : Q \times (\Sigma_1 \times \Sigma_2 \times ... \times \Sigma_n) \rightarrow 2^{Q \times (\Sigma_1 \times \Sigma_2 \times ... \times \Sigma_n) \times \{E,D\}}$$

Convencionando-se que cada elemento do conjunto $\Sigma_1 \times \Sigma_2 \times ... \times \Sigma_n$ seja representado por um novo e único símbolo, pertencente a um novo alfabeto Σ, então a função δ torna-se:

$$\delta : Q \times \Sigma \rightarrow 2^{Q \times \Sigma \times \{E,D\}}$$

idêntica, portanto, à que foi definida originalmente para a Máquina de Turing com fita limitada. Em outras palavras, a possibilidade de se convencionar que a fita de trabalho seja "repartida" em diversas trilhas não confere maior poder de reconhecimento ao dispositivo. Por outro lado, permite que o mesmo seja considerado de um outro ponto de vista, o que pode facilitar certas demonstrações, como, por exemplo, a do Teorema 5.4 apresentado a seguir.

Teorema 5.4 (Máquina de Turing $\Leftarrow$ gramática) *Seja $L = (G)$, com G uma gramática sensível ao contexto. Então $L = L(M)$, sendo M uma Máquina de Turing com fita limitada.*

Justificativa Assim como foi feito no Teorema 5.3, será apresentado como prova deste teorema um algoritmo que permite efetuar um mapeamento direto entre G e sua correspondente Máquina de Turing com fita limitada. Três propriedades das Máquinas de Turing com fita limitada são fundamentais para a apresentação e o entendimento do algoritmo:

i) O não-determinismo inerente à sua definição;

ii) A característica monotônica das gramáticas sensíveis ao contexto (suas formas sentenciais nunca sofrem contração após a aplicação de uma regra);

iii) A possibilidade de se considerar a fita de trabalho como sendo composta por várias trilhas, sem perda de generalidade.

Algoritmo 5.2 (Máquina de Turing ⇐ gramática) *Obtenção de uma Máquina de Turing com fita limitada que aceita a linguagem gerada por uma gramática sensível ao contexto.*

- Entrada: uma gramática sensível ao contexto G;
- Saída: uma Máquina de Turing com fita limitada M tal que $L(M) = L(G)$;
- Método:

  A máquina M que reconhece $L(G)$ apresenta uma fita de trabalho formada por duas trilhas de mesmo comprimento:

  - Na primeira trilha está inicialmente gravada a cadeia w que se deseja examinar para determinar se ela pertence ou não a L;
  - Na segunda trilha está inicialmente gravado, na primeira posição, o símbolo não-terminal S, a raiz da gramática G. A cadeia gravada na segunda trilha será denotada por z, e, portanto, inicialmente $|z| = |S| = 1$.

  A operação de M é não-determinística e acontece da seguinte forma:

  1. Determinar todos os pares (i, α) tais que, a partir da posição i da cadeia z, seja possível identificar uma subcadeia α, tal que $\alpha \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n \in P$, sendo P o conjunto das regras de G;
  2. Para cada par (i, α) obtido no passo (1), criar um novo "thread" para M, e deslocar o respectivo cursor de acesso simultaneamente sobre as duas trilhas, até a posição i da fita de trabalho; encerrar o "thread" original na configuração de rejeição;
  3. Para cada "thread" criado no passo (2):

     a) Criar n novos "threads" para M, sendo que cada "thread" j, $1 \leq j \leq n$, deve substituir, na cadeia z, a partir da posição i, a subcadeia α pela subcadeia β_j; encerrar o "thread" original na configuração de rejeição;

     b) Para cada "thread" criado no passo (3.a):

        i. Se $|z| > |w|$, encerrar o "thread" na configuração de rejeição;
        ii. Se $|z| < |w|$, desviar para o passo (1);
        iii. Se $|z| = |w|$ e $z \neq w$, desviar para o passo (1);

iv. Se $\left|z\right| = \left|w\right|$ e $z = w$, encerrar o "thread" na configuração de aceitação.

■

O Algoritmo 5.2 faz com que M reproduza, na segunda trilha, as formas sentenciais de G na síntese da cadeia fornecida (caso esta seja uma sentença de L). Cada uma das diferentes seqüências de derivação possíveis de serem obtidas em G são simuladas por um "thread" distinto de M, que opera de forma não-determinística.

Dessa forma, a cadeia de entrada será aceita se e apenas se existir pelo menos um "thread" que simule a seqüência de derivações que gera w em G e, portanto, que conduza M desde a sua configuração inicial até uma configuração de aceitação. Se todas as seqüências possíveis tiverem sido consideradas (através do não-determinismo) e ainda assim nenhuma delas tiver conduzido M a uma configuração de aceitação, então certamente a cadeia de entrada não pertence a L. Do ponto de vista gramatical, isso corresponde ao fato de não existir derivação possível para a referida cadeia em G. Logo, M é capaz de aceitar apenas e tão somente as sentenças que são geradas por G.

É possível provar, em função do caráter monotônico das derivações obtidas através das gramáticas sensíveis ao contexto, que todos os "threads" criados pelo Algoritmo 5.2 se encerram após um tempo finito de execução, correspondente ao tempo necessário para a geração de uma forma sentencial de comprimento mínimo igual ao da cadeia de entrada. Assim, garante-se a existência de uma Máquina de Turing com fita limitada que sempre pára, qualquer que seja a cadeia de entrada que lhe seja submetida. Portanto, $L(G)$ é uma linguagem recursiva.

O Algoritmo 5.2 presume que leitor saiba como realizar as seguintes operações fundamentais em uma Máquina de Turing com fita limitada —que, por serem extensas, embora não necessariamente complexas, serão deixadas apenas como exercício:

- Deslocar o cursor de acesso até uma posição arbitrária da fita de trabalho;
- Reconhecer um padrão (cadeia) na fita de trabalho, a partir da posição corrente do cursor de acesso;
- Deslocar parte da cadeia de entrada para a direita, com a finalidade de abrir espaço para a inserção dos símbolos indicados pela produção que estiver sendo aplicada;
- Substituir uma subcadeia localizada em uma das trilhas por uma subcadeia correspondente, conforme indicado em alguma das produções da gramática;
- Determinar se a cadeia a ser reescrita na segunda trilha eventualmente necessita de mais posições livres do que a fita de trabalho dispõe;
- Comparar as cadeias contidas nas duas trilhas da fita de trabalho para determinar se são ou não idênticas.

Exemplo 5.19 Seja $G = (\{a, b, c, Q, S\}, \{a, b, c\}, \{S \rightarrow abc \mid aSQ, bQc \rightarrow bbcc, cQ \rightarrow Qc\}, S)$, $L(G) = \{a^n b^n c^n, n \geq 1\}$. A Figura 5.7 reproduz uma seqüência de movimentos de M que conduz à aceitação da cadeia "$aabbcc$". São mostrados os conteúdos das duas trilhas da fita de trabalho.

Passo	Configuração da fita de trabalho com suas duas trilhas e os respectivos cursores de acesso	Comentário
1	1 2 3 4 5 6 7 8	Os números indicam as posições da fita de entrada.

2	< a a b b c c >
	  |
	< S >
	  |
		Configuração inicial: cursor de acesso na posição 2.

3	< a a b b c c >
	  |
	< S >
	  |
		A própria posição 2 é selecionada arbitrariamente.

4	< a a b b c c > | < a S Q > |	A subcadeia S é identificada na posição 2 da segunda trilha e a regra S → aSQ é aplicada, alterando o conteúdo da mesma.
5	< a a b b c c > | < a S Q > |	A posição 3 é selecionada arbitrariamente e o cursor de acesso se desloca até ela.
6	< a a b b c c > | < a a b c Q > |	A subcadeia S é identificada na posição 3 da segunda trilha e a regra S → abc é aplicada, alterando o conteúdo da mesma.
7	< a a b b c c > | < a a b c Q > |	A posição 5 é selecionada arbitrariamente e o cursor de acesso se desloca até ela.
8	< a a b b c c > | < a a b Q c > |	A subcadeia cQ é identificada na posição 5 da segunda trilha e a regra cQ → Qc é aplicada, alterando o conteúdo da mesma.
9	< a a b b c c > | < a a b Q c > |	A posição 4 é selecionada arbitrariamente e o cursor de acesso se desloca até ela.

10	< a a b b c c >
	  |
	< a a b b c c >
	  |
		A subcadeia bQc é identificada na posição 4 da segunda trilha e a regra bQc → bbcc é aplicada, alterando o conteúdo da mesma.

11	-	As cadeias gravadas nas suas trilhas são idênticas e M pára em configuração final, aceitando "aabbcc" como pertencente a L.

Figura 5.7: Máquina de Turing com fita limitada simulando gramática sensível ao contexto

Assim como foi feito no Exemplo 5.18, por uma questão de economia de espaço, são mostrados aqui apenas os movimentos realizados por M que permitem o reconhecimento da cadeia de entrada, omitindo-se as múltiplas tentativas de reconhecimento através do não-determinismo inerente ao dispositivo. Trata-se, portanto, de uma situação de melhor caso, deixando-se para o leitor, como exercício, a investigação dos demais caminhos que não levam ao reconhecimento da sentença.

Suponha-se, no entanto, que no passo 4 a regra escolhida tivesse sido $S \rightarrow abc$ (no lugar de $S \rightarrow aSQ$). Neste caso, a forma sentencial resultante seria "abc", portanto diferente da cadeia "$aabbcc$" gravada na primeira trilha, e no passo seguinte não seria possível identificar o lado esquerdo de nenhuma regra em "abc", disso resultando a parada de M em uma configuração não-final.

Suponha-se, por outro lado, que no passo 5 a posição 2 tivesse sido selecionada (e não a posição 3). Neste caso, M não seria capaz de reconhecer o lado esquerdo de nenhuma regra a

partir da referida posição 2, pois não existem regras que possuam o lado esquerdo começando com o símbolo "a", e portanto M pararia em uma configuração não-final.

Essas duas possibilidades, no entanto, não indicam que a cadeia de entrada possa ser arbitrariamente aceita ou rejeitada. É preciso lembrar que se trata de um dispositivo não-determinístico, no qual todas as alternativas de movimentação devem ser consideradas, até que pelo menos uma resulte em aceitação (como é o caso mostrado na Figura 5.7 do Exemplo 5.19), ou então que todas tenham resultado em parada em configuração não-final, caso em que a cadeia de entrada é rejeitada.

A rejeição, no entanto, não aconteceria no Exemplo 5.19, mesmo que as hipóteses consideradas nos parágrafos anteriores tivessem de fato acontecido durante a operação de M, pois a Figura 5.7 comprova que existe pelo menos uma seqüência de movimentos que conduz M de sua configuração inicial até uma configuração final. □

5.6 Relação entre Linguagens Sensíveis ao Contexto e Linguagens Livres de Contexto

Nesta seção será investigada a relação existente entre as linguagens livres de contexto, estudadas no Capítulo 4, e as linguagens sensíveis ao contexto, apresentadas no presente capítulo.

Teorema 5.5 (Livres de contexto $\subseteq$ sensíveis ao contexto) *Toda linguagem livre de contexto L é também uma linguagem sensível ao contexto.*

Justificativa Se L é livre de contexto, então existe pelo menos uma gramática livre de contexto G que gera L. Conforme demonstrado no Capítulo 4, toda gramática livre de contexto pode ser colocada na Forma Normal de Chomsky, com as regras no formato $A \rightarrow BC$ ou $A \rightarrow a$, com $A, B, C \in N$, $a \in \Sigma$. Se $\epsilon \in L$, então $S \rightarrow \epsilon$ é admitida como única regra vazia em G.

Se $\epsilon \in L$, então $L - \{\epsilon\}$ pode ser gerada por uma gramática livre de contexto que, além de estar na Forma Normal de Chomsky, é claramente uma gramática sensível ao contexto também. Se $\epsilon \notin L$, idem. Logo, L é também uma linguagem sensível ao contexto. ■

Exemplo 5.20 Sejam $G_1 = (\{a, b, S\}, \{S\}, \{S \rightarrow aSb \mid \epsilon\}, S)$ e $L(G_1) = \{a^n b^n, n \geq 0\}$. G_1 é uma gramática livre de contexto e, portanto, $L(G_1)$ é uma linguagem livre de contexto. Seja agora $G_2 = (\{a, b, S, X\}, \{S, X\}, P_2, S)$, com P_2 :

$$\begin{array}{rcl} \{S & \rightarrow & \epsilon \\ A & \rightarrow & a \\ B & \rightarrow & b \\ S & \rightarrow & AX \\ X & \rightarrow & SB\} \end{array}$$

É fácil notar que $L(G_2) = L(G_1)$, sendo que G_2 está na Forma Normal de Chomsky. Como se pode observar, $L - \{\epsilon\}$ é gerada por $G_3 = (\{a, b, S, X\}, \{S, X\}, P_3, S)$, em que $P_3 = P_2 - \{S \rightarrow \epsilon\}$. Portanto, G_3 é uma gramática sensível ao contexto. Logo, L é também uma linguagem sensível ao contexto. □

Teorema 5.6 (Livres de contexto $\neq$ sensíveis ao contexto) *A classe das linguagens livres de contexto constitui subconjunto próprio da classe das linguagens sensíveis ao contexto.*

Justificativa Através da aplicação do "Pumping Lemma" para linguagens livres de contexto é possível provar que diversas linguagens não são livres de contexto. Entre estas,

pode-se citar a linguagem $\{a^n b^n c^n \mid n \geq 1\}$ (ver Exemplo 4.38), a qual, no entanto, pode ser representada através de uma gramática sensível ao contexto (ver Exemplo 5.3). Logo, trata-se de uma linguagem sensível ao contexto, não-livre de contexto, e sua simples existência demonstra o teorema. ■

5.7 Linguagens que não são Sensíveis ao Contexto

É possível demonstrar que existem linguagens que não pertencem à classe das linguagens sensíveis ao contexto (Teorema 5.8). Tal demonstração, no entanto, depende da demonstração prévia de que o conjunto das gramáticas sensíveis ao contexto é enumerável (Teorema 5.7). O Capítulo 6 discutirá uma importante classe de linguagens que inclui as linguagens sensíveis ao contexto como subconjunto próprio.

Teorema 5.7 (Gramáticas sensíveis ao contexto ⇒ enumerável) *O conjunto das gramáticas sensíveis ao contexto sobre um certo alfabeto* Σ *é enumerável.*

Justificativa Considere-se um alfabeto Σ qualquer e todas as gramáticas sensíveis ao contexto que podem ser criadas a partir do mesmo. Seja $G = \{G_1, G_2, G_3, ...G_n, ...\}$ o conjunto infinito que reúne todas essas gramáticas.

Sem perda de generalidade, pode-se considerar que os símbolos não-terminais de cada uma dessas gramáticas pertencem ao conjunto infinito $A = \{A_1, A_2, A_3, ...A_n, ...\}$ e, além disso, que a raiz de cada uma delas é A_1. Ou seja, todas elas têm a mesma raiz e compartilham o mesmo conjunto de símbolos não-terminais.

Os elementos de G podem ser listados em uma ordem $G_1, G_2, G_3, ...$ (ou seja, enumerados) de acordo com o método descrito no Algoritmo 5.3.

Algoritmo 5.3 (Gramáticas sensíveis ao contexto ⇒ enumerável) *Enumeração de todas as gramáticas sensíveis ao contexto.*

- Entrada: um alfabeto Σ;
- Saída: uma enumeração de todas as gramáticas sensíveis ao contexto sobre Σ;
- Método:

    1. Inicialmente, listam-se todas as gramáticas cujas regras tenham o formato $\alpha \rightarrow \beta, |\alpha| = 1, |\beta| = 1$, e tais que apenas o não-terminal A_1 seja utilizado nas mesmas. Claramente, existe apenas um número finito de gramáticas nesta condição.

    2. A seguir, listam-se todas as gramáticas cujas regras tenham o formato $\alpha \rightarrow \beta, |\alpha| \leq 2, |\beta| \leq 2$, e tais que apenas os não-terminais A_1 e A_2 sejam utilizados nas mesmas. Novamente, o conjunto de gramáticas que satisfaz a esta condição é finito.

    3. Repete-se o passo (2) considerando $|\alpha| \leq 3, |\beta| \leq 3$ e os não-terminais A_1, A_2, A_3, e assim por diante.

Todas as gramáticas pertencentes a G serão inevitavelmente listadas (enumeradas) por este método, que gera uma seqüência infinita de conjuntos finitos, seqüência esta que

pode ser ordenada na mesma seqüência dos números naturais $(1, 2, 3...)$. O conjunto das gramáticas sensíveis ao contexto é, portanto, um conjunto enumerável. ■

Exemplo 5.21 Considere-se $\Sigma = \{a, b\}$ e, inicialmente, apenas as regras em que $\alpha \rightarrow \beta, \left|\alpha\right| = 1, \left|\beta\right| = 1$ e tais que A_1 seja o único não-terminal empregado. Então, $N_1 = \{A_1\}$, $V_1 = \Sigma \cup N_1 = \{A_1, a, b\}$. Tem-se, portanto, $m = \left|V_1\right| = 3$ regras distintas nesta categoria:

1. $A_1 \rightarrow A_1$
2. $A_1 \rightarrow a$
3. $A_1 \rightarrow b$

Desta forma, as diversas gramáticas que podem ser criadas a partir das regras acima conterão, respectivamente, apenas uma, apenas duas ou apenas três regras distintas.

Gramáticas com exatamente uma regra:

- G_1, com a regra (1)
- G_2, com a regra (2)
- G_3, com a regra (3)

Gramáticas com exatamente duas regras:

- G_4, com as regras (1) e (2)
- G_5, com as regras (2) e (3)
- G_6, com as regras (1) e (3)

Gramáticas com exatamente três regras:

- G_7, com as regras (1), (2) e (3)

Portanto, são sete as gramáticas sensíveis ao contexto distintas que podem ser criadas sobre o vocabulário $\{A_1, a, b\}$.

Para relacionar as regras em que $\alpha \rightarrow \beta, \left|\alpha\right| \leq 2, \left|\beta\right| \leq 2$, e tais que A_1, A_2 sejam empregados, deve-se considerar os seguintes casos:

a) $\left|\alpha\right| = \left|\beta\right| = 1$

b) $\left|\alpha\right| = 1, \left|\beta\right| = 2$

c) $\left|\alpha\right| = \left|\beta\right| = 2$

Sejam, portanto, $N_2 = \{A_1, A_2\}$ e $V_2 = \{A_1, A_2, a, b\}$. Então, as seguintes regras podem ser geradas em cada caso:

a) $\alpha \rightarrow \beta, \left|\alpha\right| = 1, \left|\beta\right| = 1$

$$\begin{array}{llll} A_1 \rightarrow A_1, & A_1 \rightarrow A_2, & A_1 \rightarrow a, & A_1 \rightarrow b, \\ A_2 \rightarrow A_1, & A_2 \rightarrow A_2, & A_2 \rightarrow a, & A_2 \rightarrow b \end{array}$$

Tem-se, portanto, $m = \left|N_2\right| * \left|V_2\right| = 2 * 4 = 8$ produções distintas nesta categoria.

b) $\alpha \rightarrow \beta, \left|\alpha\right| = 1, \left|\beta\right| = 2$

$A_1 \to A_1A_1$, $A_1 \to A_1A_2$, $A_1 \to A_1a$, $A_1 \to A_1b$,
$A_1 \to A_2A_1$, $A_1 \to A_2A_2$, $A_1 \to A_2a$, $A_1 \to A_2b$,
$A_1 \to aA_1$, $A_1 \to aA_2$, $A_1 \to aa$, $A_1 \to ab$,
$A_1 \to bA_1$, $A_1 \to bA_2$, $A_1 \to ba$, $A_1 \to bb$,
$A_2 \to A_1A_1$, $A_2 \to A_1A_2$, $A_2 \to A_1a$, $A_2 \to A_1b$,
$A_2 \to A_2A_1$, $A_2 \to A_2A_2$, $A_2 \to A_2a$, $A_2 \to A_2b$,
$A_2 \to aA_1$, $A_2 \to aA_2$, $A_2 \to aa$, $A_2 \to ab$,
$A_2 \to bA_1$, $A_2 \to bA_2$, $A_2 \to ba$, $A_2 \to bb$

ou seja:

- $m = \left|N_2\right| = 2$ possibilidades distintas para cadeia α, e
- $n = \left|V_2\right|^{\left|\beta\right|} = 2^4 = 16$ possibilidades distintas para a cadeia β.

Portanto, existem $m * n = 32$ produções distintas nesta categoria.

c) $\alpha \to \beta, \left|\alpha\right| = 2, \left|\beta\right| = 2$

- $m = \left|V_2\right|^2 - \left|\Sigma\right|^2 = 4^2 - 2^2 = 12$
- $n = \left|V_2\right|^{\left|\beta\right|} = 4^2 = 16$

Conseqüentemente, existem $m * n = 12 * 16 = 192$ produções distintas nesta categoria.

Consideradas como um todo, são $8 + 32 + 192 = 232$ produções distintas que satisfazem à condição $\alpha \to \beta, \left|\alpha\right| \leq 2, \left|\beta\right| \leq 2$. Com essas produções, é possível criar $2^{232} - 1$ gramáticas distintas.

Cumpre notar que o cálculo de m é feito considerando-se o total de cadeias de comprimento 2 possíveis de serem construídas sobre $V - \left|V_2\right|^2$, descontando-se, destas, aquelas que sejam formadas exclusivamente por símbolos terminais — ou seja, $\left|\Sigma\right|^2$. Dessa forma, são contadas apenas as cadeias α que contenham pelo menos um símbolo não-terminal.

A generalização desses casos pode ser resumida na questão: quantas gramáticas sensíveis ao contexto distintas podem ser geradas com n símbolos não-terminais distintos e com as regras obedecendo ao formato $\alpha \to \beta, \left|\alpha\right| \leq n, \left|\beta\right| \leq n, \left|\alpha\right| \leq \left|\beta\right|$?

Para responder a essa pergunta, deve-se, inicialmente, considerar todos os formatos diferentes que as regras podem assumir, de acordo com tal especificação. Esses formatos são apresentados na Tabela 5.1:

1::1					
1::2	2::2				
1::3	2::3	3::3			
...	...	...			
1::n-1	2::n-1	3::n-1	...	n-1::n-1	
1::n	2::n	3::n	...	n-1::n	n::n

Tabela 5.1: Cálculo do número de gramáticas sensíveis ao contexto

Nesta tabela, $i :: j$ denota o conjunto das regras $\alpha \to \beta, \left|\alpha\right| = i, \left|\beta\right| = j$. A união de todos esses conjuntos resulta no conjunto de regras $\alpha \to \beta, \left|\alpha\right| \leq n, \left|\beta\right| \leq n, \left|\alpha\right| \leq \left|\beta\right|$. Como cada um dos conjuntos considerados é finito, o conjunto resultante também será finito.

A título de ilustração, é fácil deduzir que a quantidade total de elementos contidos no conjunto $i :: j$ pode ser obtida pelo produto $m * n$, onde:

- $m = \left| V_n \right|^i - \left| \Sigma \right|^i$
  (quantidade de cadeias α distintas com comprimento i)
- $n = \left| V_n \right|^j$
  (quantidade de cadeias β distintas com comprimento j)

Para se determinar a quantidade total de regras na tabela, basta calcular a função $regras(n)$ abaixo (V_n é o conjunto dos símbolos não-terminais e Σ o conjunto dos símbolos terminais):

$regras(n)$:

1. regras $\leftarrow$ 0;
2. para $i \leftarrow$ 1 até n

   para $j \leftarrow i$ até n

   faça regras $\leftarrow$ regras+$(\left| V_n \right|^i - \left| \Sigma \right|^i) * \left| V_n \right|^j$

Como $\Sigma = \{a, b\}$, então $\left|\Sigma\right| = 2$ e $\left| V_n \right| = n + 2$. Logo:

$regras(n)$:

1. regras $\leftarrow$ 0;
2. para $i \leftarrow$ 1 até n

   para $j \leftarrow i$ até n

   faça regras $\leftarrow$ regras+$((n+2)^i - 2^i) * (n+2)^j$

Os casos anteriores podem ser considerados casos particulares da fórmula geral acima. Considere-se $n = 2, \alpha \rightarrow \beta, \left|\alpha\right| \leq 2, \left|\beta\right| \leq 2$:

$$
\begin{array}{ll}
 & regras = 0 \\
i = 1, j = 1 & regras = regras + ((2+2)^1 - 2^1) * (2+2)^1 = regras + 8 \\
i = 1, j = 2 & regras = regras + ((2+2)^1 - 2^1) * (2+2)^2 = regras + 32 \\
i = 2, j = 2 & regras = regras + ((2+2)^2 - 2^2) * (2+2)^2 = regras + 192
\end{array}
$$

Portanto, obtêm-se 8+32+192=232 regras distintas.
A quantidade total de gramáticas que podem ser criadas a partir de k regras distintas é dada pela função $gramaticas(k)$ abaixo:

$$gramaticas(k) = 2^k - 1$$

No presente exemplo, tem-se que $\alpha \rightarrow \beta, \left|\alpha\right| \leq 2, \left|\beta\right| \leq 2$ produz $2^{232} - 1$ gramáticas distintas, exatamente o mesmo resultado obtido anteriormente.
O processo todo pode, portanto, ser resumido nos seguintes passos:

1. $n \leftarrow 1$;

2. $k \leftarrow regras(n)$;
3. calcula $gramaticas(k)$;
4. $n \leftarrow n + 1$;
5. Desviar para (1).

Naturalmente, esses passos servem apenas como ilustração da forma como se podem enumerar todas as gramáticas possíveis sobre $\{a, b\}$. A sua transcrição para a forma de um algoritmo só faz sentido caso se estabeleça um limite superior para n, caso contrário eles não se encerrariam nunca.

Cumpre notar que o método descrito gera toda e qualquer gramática sensível ao contexto sobre $\{a, b\}$, incluindo aquelas que geram linguagens vazias, contêm símbolos inúteis ou são equivalentes a outras gramáticas geradas anteriormente. Como exemplo destes casos, pode-se considerar as gramáticas:

- $G_1 = (\{A_1, a, b\}, \{a, b\}, \{A_1 \rightarrow A_1\}, A_1)$
- $G_2 = (\{A_1, A_2, a, b\}, \{a, b\}, \{A_1 \rightarrow A_2, A_3 \rightarrow A_3, A_2 \rightarrow a\}, A_1)$
- $G_3 = (\{A_1, A_2, a, b\}, \{a, b\}, \{A_1 \rightarrow A_2, A_2 \rightarrow A_2, A_2 \rightarrow a\}, A_1)$

$L(G_1) = \varnothing$, A_3 é inútil em G_2 e $L(G_3) = L(G_1)$. Tal aspecto não invalida, no entanto, o prosseguimento da demonstração. □

Teorema 5.8 (Linguagens não-sensíveis ao contexto) *Existem linguagens que não são sensíveis ao contexto.*

Justificativa É feita provando-se que existe pelo menos uma linguagem tal que não existe gramática sensível ao contexto que a gere.

Considere-se inicialmente o resultado do Teorema 5.7 (as gramáticas sensíveis ao contexto são enumeráveis) e um alfabeto Σ qualquer. Em seguida, pode-se considerar uma enumeração, em ordem lexicográfica crescente, das cadeias de Σ^+. Considere-se, em particular, $\Sigma = \{a, b\}$. Tal enumeração é correspondente à seqüência:

$$a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, ...$$

Tendo enumerado os elementos de $G = \{G_1, G_2, ..., G_n, ...\}$ e de $\Sigma^+ = \{\alpha_1, \alpha_2, ..., \alpha_n, ...\}$, passa-se agora a estabelecer uma função bijetora entre os dois conjuntos, de tal forma que os pares $(G_i, \alpha_i), i \geq 1$ sejam elementos dessa função (Tabela 5.2):

G_1	G_2	G_3	...	G_n	...
$\updownarrow$	$\updownarrow$	$\updownarrow$		$\updownarrow$	
α_1	α_2	α_3	...	α_n	...

Tabela 5.2: Função bijetora entre gramáticas e cadeias sobre $\{a, b\}$

Defina-se agora a linguagem $L_R = \{\alpha_i \mid \alpha_i \notin L(G_i), \forall\, i \geq 1\}$. Em outras palavras, pertencem a L_R as cadeias α_i que não pertençam a $L(G_i)$. Tal verificação pode sempre ser feita para gramáticas sensíveis ao contexto, conforme demonstrado em teorema anterior sobre a equivalência deste tipo de gramáticas com as Máquinas de Turing com fita limitada (Teorema 5.4).

Só existem duas hipóteses acerca de L_R: ou trata-se de uma linguagem que é sensível ao contexto, ou então trata-se de uma linguagem que não é sensível ao contexto.

Considere-se que L_R seja uma linguagem sensível ao contexto. Se essa hipótese for verdadeira, deverá existir pelo menos uma gramática sensível ao contexto que a gere. Naturalmente, tal gramática deverá pertencer ao conjunto G. Seja G_i esta gramática. Se $L_R = L(G_i)$, então só existem duas possibilidades: α_i pertence ou não pertence a L_R.

- *Primeira possibilidade*: se $\alpha_i \notin L(G_i)$, então $\alpha_i \in L_R$, por hipótese, o que é uma contradição.
- *Segunda possibilidade*: se $\alpha_i \in L(G_i)$, então $\alpha_i \notin L_R$, por construção, o que também é uma contradição.

Logo, L_R não pode ser uma linguagem sensível ao contexto, e isso completa a demonstração. ■

5.8 Propriedades de Fechamento

A classe das linguagens sensíveis ao contexto é fechada em relação às operações de:

- União
- Concatenação
- Intersecção
- Complementação

porém não é fechada em relação à operação de:

- Fecho de Kleene

Os três primeiros resultados serão apresentados nos teoremas seguintes. Todos os resultados, exceto o que trata da complementação, estão demonstrados em [46]. A questão do fechamento das linguagens sensíveis ao contexto em relação à operação de complementação permaneceu sem resposta durante vários anos, até que apenas em 1988 esse resultado foi provado como sendo verdadeiro (ver [64]).

Teorema 5.9 (Fecho na união) *A classe das linguagens sensíveis ao contexto é fechada em relação à operação de união.*

Justificativa Sejam L_1 e L_2 duas linguagens sensíveis ao contexto quaisquer. Então, $L_1 = L(G_1)$ e $L_2 = L(G_2)$, com G_1 e G_2 sendo duas gramáticas sensíveis ao contexto. O Algoritmo 5.4 possibilita a obtenção de $G_3 = (V_3, \Sigma_3, P_3, S_3)$, tal que $L(G_3) = L_1 \cup L_2$.

Algoritmo 5.4 (Fecho na união) *Obtenção de uma gramática sensível ao contexto que descreve a união de duas linguagens definidas por gramáticas sensíveis ao contexto.*

- Entrada: $G_1 = (V_1, \Sigma_1, P_1, S_1)$ e $G_2 = (V_2, \Sigma_2, P_2, S_2)$, duas gramáticas sensíveis ao contexto;
- Saída: uma gramática sensível ao contexto G_3, tal que $L(G_3) = L(G_1) \cup L(G_2)$;
- Método:

1. Renomear adequadamente os símbolos não-terminais de G_1 e G_2, de tal modo que $N_1 \cap N_2 = \varnothing$;
2. $V_3 = V_1 \cup V_2$;
3. $\Sigma_3 = \Sigma_1 \cup \Sigma_2$;
4. $P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_2, S_3 \rightarrow S_1)\}$.

A gramática G_3 obtida pela aplicação do Algoritmo 5.4 é sensível ao contexto, logo $L_1 \cup L_2$ é uma linguagem sensível ao contexto. ■

Teorema 5.10 (Fecho na concatenação) *As linguagens sensíveis ao contexto são fechadas em relação à operação de concatenação.*

Justificativa Sejam L_1 e L_2 duas linguagens sensíveis ao contexto quaisquer. Então, L_1 e L_2 são reconhecidas, respectivamente, por Máquinas de Turing com fita limitada M_1 e M_2. A linguagem $L_3 = L_1 L_2$ é aceita por M_3, uma Máquina de Turing com fita limitada, construída conforme o Algoritmo 5.5.

Algoritmo 5.5 (Fecho na concatenação) *Obtenção de uma Máquina de Turing com fita limitada que reconhece a concatenação de duas linguagens sensíveis ao contexto definidas por Máquinas de Turing com fita limitada.*

- Entrada: M_1 e M_2, duas Máquinas de Turing com fita limitada;
- Saída: uma Máquina de Turing com fita limitada M_3, tal que $L(M_3) = L(M_1)$ $L(M_2)$;
- Método:

  1. M_3 inicialmente simula M_1 com uma cadeia de entrada w, $w = \alpha\beta$;
  2. Se M_1 aceita um prefixo α de w, então M_3 simula M_2 com o sufixo β;
  3. Se M_2 aceita β, então M_3 pára e aceita a cadeia w.

No Algoritmo 5.5, M_3 aceita w se e somente se $\alpha \in L_1$ e $\beta \in L_2$. Caso contrário, M_3 rejeita w. Logo, L_3 é uma linguagem sensível ao contexto. ■

Teorema 5.11 (Fecho na intersecção) *As linguagens sensíveis ao contexto são fechadas em relação à operação de intersecção.*

Justificativa Sejam L_1 e L_2 duas linguagens sensíveis ao contexto quaisquer. Então, L_1 e L_2 são reconhecidas, respectivamente, por Máquinas de Turing com fita limitada M_1 e M_2. A linguagem $L_3 = L_1 \cap L_2$ é aceita por M_3, uma Máquina de Turing com fita limitada, construída conforme o Algoritmo 5.6.

Algoritmo 5.6 (Fecho na intersecção) *Obtenção de uma Máquina de Turing com fita limitada que reconhece a intersecção de duas linguagens sensíveis ao contexto definidas por Máquinas de Turing com fita limitada.*

- Entrada: M_1 e M_2, duas Máquinas de Turing com fita limitada;
- Saída: uma Máquina de Turing com fita limitada M_3, tal que $L(M_3) = L(M_1) \cap L(M_2)$;
- Método:

  1. M_3 possui duas fitas de entrada;
  2. Uma cadeia w é gravada na primeira fita;
  3. M_3 copia w da primeira para a segunda fita;
  4. M_3 simula M_1, utilizando a primeira fita;
  5. Se M_1 aceita w, então M_3 simula M_2 com a segunda fita;
  6. Se M_2 aceita w, então M_3 pára e aceita w.

De acordo com o Algoritmo 5.6, M_3 aceita w se e somente se $w \in L_1 \cap L_2$. Caso contrário, M_3 rejeita w. Logo, L_3 é uma linguagem sensível ao contexto. ■

5.9 Questões Decidíveis e Não-Decidíveis

Diversos problemas de decisão costumam ser considerados neste tipo de estudo. Em particular, dadas cadeias arbitrárias e linguagens sensíveis ao contexto quaisquer, os problemas mais comumente considerados são:

- $w \in L$?
- $L = \varnothing$?
- $L = \Sigma^*$?
- $L_1 = L_2$?
- $L_1 \subseteq L_2$?
- $L_1 \cap L_2 = \varnothing$?

De todos estes, apenas o primeiro é um problema que pode ser decidido no caso geral, quaisquer que sejam w e L. Os demais constituem problemas para os quais pode não haver solução geral. Isso significa que determinadas combinações de dados de entrada podem nunca gerar uma resposta, não importa se afirmativa ou negativa. Será apresentado aqui apenas o primeiro desses resultados, sendo que a demonstração de todos eles pode ser encontrada em [46].

Teorema 5.12 (A cadeia pertence?) *Dadas uma cadeia $w \in \Sigma^*$ e uma linguagem sensível ao contexto L, $L \subseteq \Sigma^*$, é sempre possível determinar se $w \in L$.*

Justificativa Se L é uma linguagem sensível ao contexto, então $L = L(G)$, sendo G uma gramática sensível ao contexto. Constrói-se uma Máquina de Turing com fita limitada M, composta por três trilhas:

- Na primeira trilha é gravada a cadeia de entrada w;
- Na segunda trilha é gravada uma representação de G;
- Na terceira trilha, é gravado inicialmente o símbolo inicial de G (a raiz S).

M opera de maneira muito semelhante ao modelo discutido na demonstração do Teorema 5.4. A diferença é que, naquele caso, a especificação da gramática G estava "gravada" na própria definição da Máquina de Turing com fita limitada. Agora, G está gravada em uma trilha adicional, reservada para essa finalidade.

Assim, M procura, de maneira não-determinística, gerar a cadeia w usando as regras de G gravadas na segunda trilha e construindo formas sentenciais intermediárias na terceira trilha. Através da comparação dos conteúdos da primeira e da terceira trilhas, conforme o método descrito no Teorema 5.4, é possível demonstrar que $w \in L(M)$ se e somente se $S \Rightarrow^* w$. ■

6 Linguagens Recursivas

Linguagens recursivas são aquelas que são aceitas por um tipo muito geral de reconhecedor: a Máquina de Turing sem limitação de memória, também conhecida como Máquina de Turing com fita infinita ou, simplesmente, Máquina de Turing.

Como o próprio nome sugere, as Máquinas de Turing com fita infinita diferem das apresentadas no capítulo anterior — as Máquinas de Turing com fita limitada — justamente pelo fato de não apresentarem restrições quanto ao comprimento da fita de trabalho. No que se refere aos demais aspectos, elas são definidas de forma muito similar.

Apesar de se tratar de uma diferença aparentemente pouco significativa, o fato de a fita de trabalho comportar infinitas posições possibilita, conforme está demonstrado no Teorema 6.3, o reconhecimento de uma categoria de linguagens mais abrangente do que as linguagens sensíveis ao contexto. Estas, por sua vez, conforme veremos mais adiante, constituem um subconjunto próprio das linguagens recursivas.

Diferentemente das linguagens sensíveis ao contexto, no entanto, não há caracterização conhecida para as linguagens recursivas em termos gramaticais. Ou seja, não é possível caracterizar essa classe de linguagens a partir da formulação de restrições ao formato das produções gramaticais gerais — procedimento adotado para definir as classes das linguagens regulares, das livres de contexto e também das sensíveis ao contexto.

Da mesma forma, não são conhecidas eventuais propriedades estruturais que possam ser consideradas para efeito de caracterização de uma linguagem como sendo recursiva. Se, por outro lado, as linguagens regulares, as livres de contexto e as sensíveis ao contexto podem ser facilmente identificadas a partir de tais propriedades — concatenação, união e fechamento de termos no caso das regulares, balanceamento de termos nas livres de contexto e vinculação de termos nas sensíveis ao contexto —, as linguagens recursivas não exibem características de natureza equivalente, que poderiam servir para identificar uma determinada linguagem como sendo estritamente dessa classe.

A única caracterização que resta, portanto, é baseada no modelo de reconhecimento. O restante deste capítulo será dedicado à formalização e à discussão das Máquinas de Turing e suas propriedades. Na seqüência, as linguagens recursivas serão definidas como sendo uma classe particular de linguagens aceita por esse tipo de dispositivo (pois as linguagens recursivas não são a única classe de linguagens por ele aceita). Finalmente, será demonstrado que toda linguagem sensível ao contexto é também recursiva e, além disso, que existe pelo menos uma linguagem recursiva que não é sensível ao contexto. Em resumo, demonstrar-se-á que a classe das linguagens sensíveis ao contexto constitui um subconjunto próprio da classe das linguagens recursivas. Linguagens que são recursivas porém não são sensíveis ao contexto são denominadas linguagens **estritamente recursivas**.

6.1 Máquinas de Turing

Máquinas de Turing (com fita ilimitada ou infinita) são dispositivos de reconhecimento de cadeias que possuem as seguintes diferenças em relação às Máquinas de Turing com fita limitada, discutidas no Capítulo 5:

1. A fita de trabalho possui comprimento infinito, sendo limitada à esquerda e sem limitação à direita.[1]

2. A cadeia de entrada é delimitada apenas à esquerda, por meio de um símbolo especial, representado por "<", que não faz parte do alfabeto de entrada. Juntamente com esse símbolo, a cadeia a ser processada é posicionada de forma tal que fique ajustada à esquerda sobre a fita. O cursor de acesso inicialmente aponta o símbolo mais à esquerda da cadeia de entrada, ou seja, o símbolo imediatamente à direita do marcador "<".[2]

3. As posições da fita à direita do último símbolo da cadeia de entrada são inicialmente preenchidas com um símbolo especial — representado por B, que indica que aquelas posições estão vagas. O símbolo B também não faz parte do alfabeto de entrada.[3]

4. O cursor de acesso pode se deslocar livremente sobre a fita de trabalho, exceto para a esquerda da primeira posição da fita. Caso isso ocorra, a computação encerra-se anormalmente.

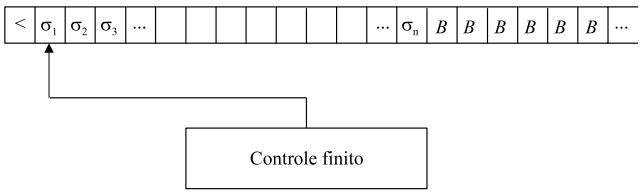

Figura 6.1: Máquina de Turing

A Figura 6.1 ilustra a configuração inicial de uma Máquina de Turing com cadeia de entrada igual a $\sigma_1\sigma_2\sigma_3...\sigma_n$. Formalmente, uma Máquina de Turing é definida como:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, <, B, F)$$

onde:

- Q é o conjunto finito não-vazio de estados;
- Σ é o alfabeto de entrada, formado por um conjunto finito não-vazio de símbolos;
- Γ é um conjunto, também finito e não-vazio, de símbolos que podem ser lidos e/ou escritos na fita de trabalho. $\Gamma \supseteq \Sigma$;
- δ é a função parcial de transição, $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{E,D\}}$;
- q_0 é o estado inicial, $q_0 \in Q$;

[1] Dependendo do autor, não há limitação tampouco à esquerda.
[2] Alguns autores desconsideram o uso do marcador "<" e supõem que a primeira posição da fita de trabalho seja ocupada pelo primeiro símbolo da cadeia a ser analisada.
[3] Dependendo do autor, outros símbolos são usados para representar posições vagas da fita de trabalho.

- "$<$" $\in \Gamma$, "$<$" $\notin \Sigma$, é o símbolo que indica a primeira posição da fita de trabalho. Durante toda a operação da máquina, o símbolo "$<$" não pode ser gravado em nenhuma outra posição da fita;

- $B \in \Gamma, B \notin \Sigma$, é o símbolo utilizado para preencher inicialmente todas as posições à direta da cadeia de entrada na fita. Durante a operação da máquina, o símbolo B pode ser gravado em qualquer posição da fita;

- $F \subseteq Q$ é o conjunto de estados finais.

A **configuração de uma Máquina de Turing** é indicada por uma tripla (α, q_k, β), onde q_k é o estado corrente, $\alpha \in \Gamma^*$ é a porção do conteúdo da fita de trabalho que se encontra à esquerda do cursor de acesso, e $\beta \in \Gamma^*$ é a porção do conteúdo da fita de trabalho que se encontra à direita do cursor de acesso, incluindo a posição correntemente apontada por ele.

A **configuração inicial** é $(<, q_0, \gamma)$, onde q_0 é o estado inicial e $\gamma \in \Sigma^*$ é a cadeia de entrada a ser analisada. O cursor de acesso aponta, portanto, exatamente o primeiro símbolo da cadeia de entrada γ. A porção α da representação (α, q_k, β) corresponde, neste caso, à cadeia unitária $<$. Uma **configuração final** é definida como (λ, q_f, μ), com $q_f \in F$ e $\lambda, \mu \in \Gamma^*$.

As transições contidas na função δ especificam possibilidades não-determinísticas de movimentação, que conduzem o dispositivo de uma configuração para a configuração seguinte. Diz-se que o dispositivo **pára** quando a função δ não for definida para o par *(estado, símbolo de entrada)* corrente.

Em caso de tentativa de deslocamento do cursor de acesso para a esquerda da primeira posição da fita, a computação encerra-se anormalmente. Neste caso, a cadeia de entrada é rejeitada, não importando se o estado em que a máquina se encontra é final ou não.

A rigor, o emprego do símbolo "$<$" para sinalizar o início da fita de trabalho é desnecessário. Diversos textos consideram que a primeira posição da fita seja ocupada pelo primeiro símbolo da cadeia de entrada, sem perda de generalidade. A incorporação do mesmo ao modelo, no entanto, contribui para a construção de máquinas mais simples, na medida em que o controle dos deslocamentos do cursor de acesso, a fim de evitar o término anormal da computação, também pode ser simplificado.

A linguagem aceita por uma Máquina de Turing com fita infinita é o conjunto das cadeias que são capazes de conduzir o dispositivo desde a sua configuração inicial (única para uma determinada cadeia de entrada) até uma configuração final qualquer na qual ele esteja parado — sem possibilidade de movimentação. Formalmente,

$$L(M) = \{\gamma \in \Sigma^* \mid (<, q_0, \gamma) \vdash^* (\lambda, q_f, \mu), q_f \in F, \lambda, \mu \in \Gamma^*\}$$

admitindo-se, como condição de parada, que $\mu = \sigma\pi, \sigma \in \Gamma, \pi \in \Gamma^*$ e δ não seja definida para (q_f, σ).

Seja L a linguagem aceita por uma Máquina de Turing M. Se $w_1 \in L$, então M pára e aceita w_1. Considere-se, porém, a cadeia $w_2 \in \Sigma^* - L$. Neste caso, M pode tanto parar, rejeitando a entrada, quanto entrar em loop infinito, sem nunca atingir uma condição de parada.

Uma linguagem L é dita **recursiva** se existir pelo menos uma Máquina de Turing M tal que:

1. Para toda cadeia $w \in L$, M pára e aceita w;

2. Para toda cadeia $z \in \Sigma^* - L$, M pára e rejeita z.

Ou seja, M atinge a condição de parada para toda e qualquer cadeia que lhe é submetida, não importando se esta pertence ou não a L.

Uma linguagem recursiva é também dita **linguagem decidível**. Este termo denota o fato de que, para essa classe de linguagens, sempre existe pelo menos uma Máquina de Turing que aceita a referida linguagem, qualquer que seja a cadeia usada como entrada (não importa se ela pertence ou não à linguagem), e que sempre pára. A parada pode ocorrer com aceitação ou rejeição da cadeia.

Conforme será discutido no Capítulo 7, as linguagens recursivas não são a única classe de linguagens aceitas pelas Máquinas de Turing. Na verdade, o relaxamento da condição de parada para cadeias não pertencentes à linguagem faz com que o modelo possa ser usado para definir uma classe mais ampla de linguagens, denominadas linguagens recursivamente enumeráveis.

Exemplo 6.1 A máquina M apresentada na Figura 6.2 é tal que $L(M) = abc(a \mid b \mid c)^*$.

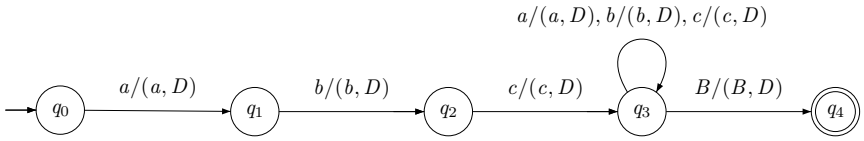

Figura 6.2: Máquina de Turing que aceita $abc(a \mid b \mid c)^*$ para o Exemplo 6.1

Os movimentos que M executa na aceitação de algumas sentenças pertencentes a L, desde a configuração inicial até a configuração final, são mostrados a seguir:

- $(<, q_0, abcB) \vdash (< a, q_1, bcB) \vdash (< ab, q_2, cB) \vdash (< abc, q_3, B) \vdash (< abcB, q_4, \beta)$
- $(<, q_0, abcaB) \vdash (< a, q_1, bcaB) \vdash (< ab, q_2, caB) \vdash (< abc, q_3, aB) \vdash$
  $(< abca, q_3, B) \vdash (< abcaB, q_4, \beta)$
- $(<, q_0, abccbaB) \vdash (< a, q_1, bccbaB) \vdash (< ab, q_2, ccbaB) \vdash (< abc, q_3, cbaB) \vdash$
  $(< abcc, q_3, baB) \vdash (< abccbB, q_3, aB) \vdash (< abccbaB, q_3, B) \vdash (< abccbaB, q_4, \beta)$

O símbolo B é usado, nestes casos, para garantir que a cadeia de entrada não contenha símbolos estranhos ao alfabeto $\{a, b, c\}$. A cadeia β, usada na configuração final dos três casos, é formada por uma seqüência infinita de símbolos B.

A rejeição de algumas cadeias pertencentes a Σ^*, porém não pertencentes a L, pode ser ilustrada através dos seguintes exemplos:

- $(<, q_0, b)$
- $(<, q_0, aB) \vdash (< a, q_1, B)$
- $(<, q_0, abb) \vdash (< a, q_1, bb) \vdash (< ab, q_2, b)$

Note-se que, nos casos acima, M pára sempre em configurações não-finais, uma vez que, respectivamente, q_0, q_1 e q_2 são estados não-finais.

L é uma linguagem recursiva, uma vez que M sempre pára, qualquer que seja a cadeia de entrada $w \in \Sigma^*$ que lhe seja apresentada. □

Exemplo 6.2 A Máquina de Turing M ilustrada na Figura 6.3 aceita a linguagem $L = \{ww^R \mid w \in \{a, b\}^*\}$.

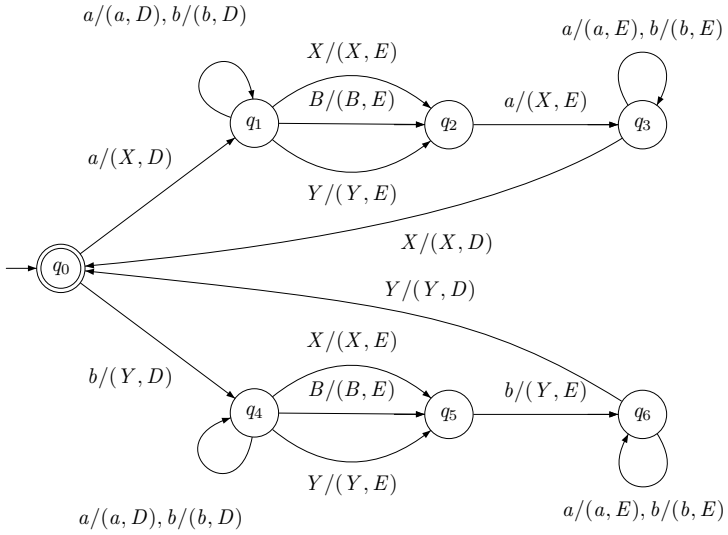

Figura 6.3: Máquina de Turing que aceita $\{ww^R \mid w \in \{a, b\}^*\}$ para o Exemplo 6.2

São exemplos de cadeias aceitas por M:

- $(<, q_0, aaB) \vdash (< X, q_1, aB) \vdash (< Xa, q_1, B) \vdash (< X, q_2, aB) \vdash$
  $(<, q_3, XXB) \vdash (< X, q_0, XB)$

- $(<, q_0, baabB) \vdash (< Y, q_4, aabB) \vdash (< Ya, q_4, abB) \vdash (< Yaa, q_4, bB) \vdash$
  $(< Yaab, q_4, B) \vdash (< Yaa, q_5, bB) \vdash (< Ya, q_6, aYB) \vdash (< Y, q_6, aaYB) \vdash$
  $(<, q_6, YaaYB) \vdash (< Y, q_0, aaYB) \vdash (< YX, q_1, aYB) \vdash (< YXa, q_1, YB) \vdash$
  $(< YX, q_2, aYB) \vdash (< Y, q_3, XXYB) \vdash (< YX, q_0, XYB)$

- $(<, q_0, bbbbbbB) \vdash (< Y, q_4, bbbbbB) \vdash (< Yb, q_4, bbbbB) \vdash$
  $(< Ybb, q_4, bbbB) \vdash (< Ybbb, q_4, bbB) \vdash (< Ybbbb, q_4, bB) \vdash$
  $(< Ybbbbb, q_4, B) \vdash (< Ybbbb, q_5, bB) \vdash (< Ybbb, q_6, bYB) \vdash$
  $(< Ybb, q_6, bbYB) \vdash (< Yb, q_6, bbbYB) \vdash (< Y, q_6, bbbbYB) \vdash$
  $(<, q_6, YbbbbYB) \vdash (< Y, q_0, bbbbYB) \vdash (< YY, q_4, bbbYB) \vdash$
  $(< YYb, q_4, bbYB) \vdash (< YYbb, q_4, bYB) \vdash (< YYbbb, q_4, YB) \vdash$
  $(< YYbb, q_5, bYB) \vdash (< YYb, q_6, bYYB) \vdash (< YY, q_6, bbYYB) \vdash$
  $(< Y, q_6, YbbYYB) \vdash (< YY, q_0, bbYYB) \vdash (< YYY, q_4, bYYB) \vdash$
  $(< YYYb, q_4, YYB) \vdash (< YYY, q_5, bYYB) \vdash (< YY, q_6, YYYYB) \vdash$
  $(< YYY, q_0, YYYB)$

São exemplos de cadeias rejeitadas por M:

- $(<, q_0, bB) \vdash (< Y, q_4, B) \vdash (<, q_5, YB)$

- $(<, q_0, abB) \vdash (< X, q_1, bB) \vdash (< Xb, q_1, B) \vdash (< X, q_2, bB)$

- $(<, q_0, bbaaaaB) \vdash (< Y, q_4, baaaaB) \vdash (< Yb, q_4, aaaaB) \vdash (< Yba, q_4, aaaB) \vdash$
  $(< Ybaa, q_4, aaB) \vdash (< Ybaaa, q_4, aB) \vdash (< Ybaaaa, q_4, B) \vdash (< Ybaaa, q_5, aB)$

□

Seja $L \subseteq \Sigma^*$ uma linguagem e M uma Máquina de Turing. Diz-se que M **aceita** L se M é capaz de atingir uma configuração final para todas as cadeias que pertencem a L, não importando o que acontece quando uma cadeia pertencente a $\Sigma^* - L$ é submetida a M. Por outro lado, diz-se que M **decide** L se, para qualquer cadeia pertencente a

Σ^*, M sempre pára, aceitando as sentenças de L e rejeitando as demais. Finalmente, M **reconhece** L se M gera saídas distintas indicando, para cada cadeia pertencente a Σ^*, se a mesma pertence ou não a L. Naturalmente, se M reconhece L, então M decide L. Se M decide L, normalmente é simples fazer M reconhecer L. Finalmente, se M decide ou reconhece L, é claro que M aceita L. A mera aceitação, por outro lado, não implica que seja possível decidir ou reconhecer uma linguagem.

Finalmente, cumpre observar que, assim como ocorre no caso dos autômatos finitos e das linguagens regulares, é possível demonstrar que toda e qualquer linguagem aceita por uma Máquina de Turing não-determinística é aceita também por pelo menos uma Máquina de Turing determinística. Tal característica difere, por outro lado, daquela exibida pelos autômatos de pilha, que, em sua versão não-determinística, são capazes de reconhecer uma classe mais ampla de linguagens do que as versões determinísticas.

6.2 Critérios de Aceitação

O critério de aceitação por estado final, utilizado nos Exemplos 6.1 e 6.2, pode ser substituído, sem prejuízo da linguagem definida por uma particular Máquina de Turing, pelo critério da parada, conforme explicado a seguir.

Seja L uma linguagem aceita por uma Máquina de Turing M_1 pelo critério de aceitação baseado em estado final. Então, se $w \in L, M_1$ inevitavelmente pára em algum estado final e aceita w. Se $w \notin L, M_1$ pára em um estado não-final e rejeita w, ou então entra em uma seqüência infinita de movimentações sem que haja qualquer evolução.

Segundo o critério da parada, se L é aceita por uma máquina M_2, então M_2 simplesmente pára, qualquer que seja a cadeia pertencente a L que lhe for submetida. Caso lhe seja submetida uma cadeia não pertencente a L, então M_2 inicia uma seqüência de movimentações que não termina. Máquinas de Turing com critério de aceitação definido segundo este princípio não apresentam estados finais.

Teorema 6.1 (Estado final $\Leftrightarrow$ parada) *Os critérios de aceitação baseados em estado final e parada, para Máquinas de Turing, são equivalentes.*

Justificativa Conforme os Algoritmos 6.1 e 6.2 apresentados a seguir.

Algoritmo 6.1 (Estado final $\Rightarrow$ parada) *Conversão de Máquina de Turing com aceitação por critério de estado final em Máquina de Turing com aceitação por critério de parada.*

- Entrada: $M = (Q, \Sigma, \Gamma, \delta, q_0, <, B, F)$ uma Máquina de Turing com critério de aceitação baseado em estado final;
- Saída: uma Máquina de Turing M' com critério de aceitação baseado em parada, tal que $L(M') = L(M)$;
- Método:

  $M' = (Q', \Sigma, \Gamma, \delta', q_0, <, B, F')$, onde:

  1. $Q' = Q \cup \{q_{loop}\}$
  2. $F' = \varnothing$

3. $\delta' = \delta$ e, além disso:

   - $\forall\, q \in Q - F, x \in \Gamma$, se $\delta(q, x)$ não é definido, então $\delta' = \delta' \cup \{((q, x), \{(q_{loop}, x, D)\})\}$
   - $\forall\, x \in \Gamma, \delta' = \delta' \cup \{((q_{loop}, x), \{(q_{loop}, x, D)\})\}$

De acordo com o Algoritmo 6.1, M' possui os mesmos estados de M, acrescido de um novo estado denominado q_{loop}. Os estados de M' são todos não-finais. As transições de M' são as mesmas de M, acrescidas das novas transições que têm o objetivo de fazer M' entrar em uma seqüência infinita de movimentações ao processar cadeias que são rejeitadas por M através da parada em um estado não-final qualquer. Cadeias que são aceitas por M provocam simplesmente a parada de M'.

Algoritmo 6.2 (Estado final $\Leftarrow$ parada) *Conversão de Máquina de Turing com aceitação por critério de parada em Máquina de Turing com aceitação por critério de estado final.*

- Entrada: $M = (Q, \Sigma, \Gamma, \delta, q_0, <, B, \varnothing)$ uma Máquina de Turing com critério de aceitação baseado em parada;
- Saída: uma Máquina de Turing M' com critério de aceitação baseado em estado final, tal que $L(M') = L(M)$;
- Método:

  $M' = (Q, \Sigma, \Gamma, \delta, q_0, <, B, F')$, onde:

  1. $F' = Q$

De acordo com o Algoritmo 6.2, os estados de M' são os mesmos de M, porém todos convertidos em estados finais. Dessa forma, toda e qualquer cadeia capaz de conduzir M a uma condição de parada necessariamente provocará a parada de M' em um estado final. ■

Exemplo 6.3 Seja a máquina M do Exemplo 6.1, cujo critério de aceitação é baseado em estado final. A máquina M', equivalente a M, porém baseada em critério de parada, é apresentada na Figura 6.4.

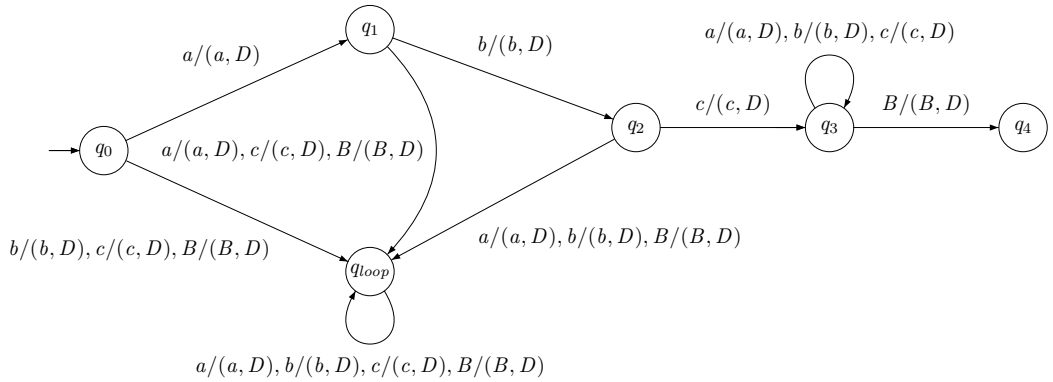

Figura 6.4: Máquina de Turing com critério de aceitação baseado em parada para o Exemplo 6.3

É fácil notar, neste exemplo, que as cadeias a, b e abb, rejeitadas após parada em M, conduzem M' a uma seqüência interminável de movimentos. Senão, vejamos:

$$\begin{aligned}
(<, q_0, abbBB...) &\vdash \\
(< a, q_1, bbBB...) &\vdash \\
(< ab, q_2, bBB...) &\vdash \\
(< abb, q_{loop}, BB...) &\vdash \\
(< abbB, q_{loop}, B...) &\vdash \\
(< abbBB, q_{loop}, ...) &\vdash \quad ...
\end{aligned}$$

Já as cadeias aceitas por estado final em M — abc, $abca$ e $abccba$ — conduzem M' a uma situação de parada, e portanto de aceitação, segundo este critério:

$$\begin{aligned}
(<, q_0, abccbaB) &\vdash \\
(< a, q_1, bccbaB) &\vdash \\
(< ab, q_2, ccbaB) &\vdash \\
(< abc, q_3, cbaB) &\vdash \\
(< abcc, q_3, baB) &\vdash \\
(< abccbB, q_3, aB) &\vdash \\
(< abccbaB, q_3, B) &\vdash (< abccbaB, q_4, \beta)
\end{aligned}$$

O caso inverso (conversão do critério de parada para o critério de estado final) é trivial e não será exemplificado. □

6.3 Extensões Mais Comuns das Máquinas de Turing

O poder computacional das Máquinas de Turing já foi questionado diversas vezes, sendo que, em muitas delas, foram feitas propostas de extensões que seriam supostamente capazes de aumentar o seu poder — ou, traduzindo em termos de linguagens, de expandir a classe de linguagens que elas são capazes de reconhecer.

Provas formais, no entanto, demonstram que o dispositivo básico da Máquina de Turing com fita infinita, formulado na Seção 6.1, e complementado na Seção 6.2, possui

exatamente o mesmo poder computacional de qualquer outra versão estendida. Entre as extensões mais populares, citamos as seguintes (nenhuma equivalência com o modelo básico será demonstrada neste texto):

- *Múltiplas trilhas*:
  Trata-se do caso discutido no texto da Seção 5.5, que precede o Teorema 5.4 (equivalência das gramáticas sensíveis ao contexto com as Máquinas de Turing com fita limitada). Tal extensão considera que a fita de trabalho seja substituída por uma coleção finita de fitas, as quais são acessadas por uma quantidade idêntica de cursores de acesso, cada qual dedicado à fita correspondente. Os cursores de acesso deslocam-se todos juntos e sempre no mesmo sentido (para a esquerda ou para a direita), sob o comando do controle finito. Os símbolos gravados em cada fita, no entanto, podem ser distintos. Exemplo: em um dispositivo com duas fitas, os dois cursores de acesso devem se movimentar simultaneamente para a esquerda ou para a direita, ainda que cada fita receba a gravação de símbolos de alfabetos distintos.

  Função de transição correspondente (para n trilhas, cada qual com o seu próprio alfabeto Σ_i):

  $$\delta : Q \times (\Sigma_1 \times \Sigma_2 ... \times \Sigma_n) \rightarrow Q \times (\Sigma_1 \times \Sigma_2 ... \times \Sigma_n) \times \{E, D\}$$

- *Múltiplas fitas*:
  Esta extensão assemelha-se à que foi discutida para o caso de múltiplas trilhas, com a diferença de que os cursores de acesso podem ser movimentados de forma independente uns dos outros. Assim, o controle finito pode especificar, na movimentação de uma configuração para a configuração seguinte, o comportamento de cada cursor de acesso em sua respectiva fita de trabalho: sentido do deslocamento e símbolo a ser gravado na respectiva fita. Por exemplo: em um dispositivo com duas fitas, uma transição pode especificar que o cursor da primeira fita se desloque para a esquerda, ao mesmo tempo que o cursor da segunda fita se desloque para a direita.

  Função de transição correspondente (para n fitas, cada qual com seu próprio alfabeto Σ_i):

  $$\delta : Q \times (\Sigma_1 \times \Sigma_2 ... \times \Sigma_n) \rightarrow Q \times (\Sigma_1 \times \{E, D\}) \times (\Sigma_2 \times \{E, D\}) ... \times (\Sigma_n \times \{E, D\})$$

- *Múltiplos cursores*:
  Neste caso, uma única fita de trabalho é equipada com dois ou mais cursores de acesso. O controle finito, por sua vez, especifica o sentido de movimentação de cada cursor e o símbolo a ser gravado na posição da fita correntemente referenciada por intermédio de cada cursor. Cuidado especial deve ser tomado com a possibilidade de dois ou mais cursores referenciarem uma mesma posição da fita de trabalho, para que sejam evitados conflitos devidos à ordem em que forem gravados ou lidos os símbolos pelos respectivos cursores de acesso.

  Função de transição correspondente (para n cursores):

  $$\delta : Q \times (\Sigma \times \Sigma ... \times \Sigma) \rightarrow Q \times (\Sigma \times \{E, D\}) \times (\Sigma \times \{E, D\}) ... \times (\Sigma \times \{E, D\})$$

- *Fita ilimitada em ambos os sentidos*:
  Originalmente concebida de forma limitada à esquerda e ilimitada à direita, esta extensão contempla que a fita de trabalho tenha um comprimento ilimitado em ambos os sentidos. Não há marcação de início de cadeia (símbolo "$<$") e o cursor pode se deslocar livremente tanto para a esquerda quanto para a direita, sem risco de encerrar a operação anormalmente, por tentativa de acesso a posições à esquerda do início da fita. A cadeia de entrada é gravada, neste caso, em qualquer posição da fita de trabalho, e o restante da mesma é preenchido com brancos (símbolo "B"). Convenciona-se, também, a posição do cursor de acesso no início da operação do dispositivo (por exemplo, apontando para o símbolo mais à esquerda da cadeia de entrada).

  A função de transição correspondente permanece inalterada em relação à definição original:
  $$\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{E, D\}$$

- *Transições que deslocam o cursor um número variável de posições*:
  Corresponde ao caso em que o cursor de acesso, após a execução de uma transição, é instruído para se deslocar um número arbitrário de posições sobre a fita de trabalho, para a direita ou para a esquerda. Este número pode inclusive ser zero, caso em que o cursor permanece na posição em que se encontrava antes da aplicação da transição, sem se deslocar para esquerda nem para a direita.

  Função de transição correspondente:
  $$\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \ (\{E\}^* \cup \{D\}^*)$$

- *Transições sem leitura ou gravação de símbolos*:
  Corresponde ao caso em que o símbolo apontado pelo cursor de acesso é irrelevante para a escolha da movimentação a ser efetuada pelo dispositivo (transição sem leitura), ou, ainda, ao caso em que, após a execução da movimentação, o conteúdo da posição apontada pelo cursor não é modificado (transição sem gravação).

  Função de transição correspondente:
  $$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow Q \times (\Sigma \cup \{\epsilon\}) \times \{E, D\}$$

As demonstrações para os quatro casos iniciais podem ser encontradas em [46]. Todas essas extensões podem, eventualmente, ser combinadas umas com as outras, gerando diferentes versões da Máquina de Turing básica. Se, por um lado, elas nada conseguem acrescentar ao poder da Máquina de Turing original, elas podem, por outro lado, facilitar significativamente o projeto, a implementação e a análise de algoritmos representados através de Máquinas de Turing, assim como a demonstração de teoremas acerca desse tipo de dispositivos ou das linguagens por eles aceitas.

Trata-se, portanto, de uma conveniência que deve ser levada em conta e que pode ser usada sem qualquer perigo de introduzir "distorções" no modelo original que possam eventualmente descaracterizar as suas propriedades fundamentais ou mesmo inviabilizar a sua realização prática.

6.4 Relação entre Linguagens Recursivas e Linguagens Sensíveis ao Contexto

A seguir, apresentam-se dois importantes resultados referentes à relação entre as linguagens recursivas e as linguagens sensíveis ao contexto. Inicialmente, é demonstrado que toda linguagem sensível ao contexto é também uma linguagem recursiva. Em seguida, mostra-se que a classe das linguagens sensíveis ao contexto constitui um subconjunto próprio da classe das linguagens recursivas.

Teorema 6.2 (Sensíveis ao contexto $\subseteq$ recursivas) *Toda linguagem sensível ao contexto é também recursiva.*

Justificativa Seja L uma linguagem sensível ao contexto. Então, existe uma Máquina de Turing com fita limitada M_1 que aceita L:

$$L = L(M_1), M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, <, >, q_0, F_1)$$

Constrói-se então uma Máquina de Turing M_2, sem limitação de tamanho para a fita de entrada, que simula M_1:

$$M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, <, B, q_0, F_2)$$

onde:

- $Q_2 = Q_1$
- $\Gamma_2 = \Gamma_1$
- $F_2 = F_1$
- $\delta_2 = \delta_1$, exceto pelo fato de que toda e qualquer transição de M_1 da forma $\delta_1(q_i, >) = (q_j, >, E)$ deve ser substituída por uma transição da forma $\delta_2(q_i, B) = (q_j, B, E)$.

A configuração inicial de M_2 será $(<, q_0, \gamma)$ quando a configuração inicial de M_1 for $(<, q_0, \gamma >)$. Como a função de transição δ é idêntica em ambas as máquinas, exceto pelas transições da forma:

$$\delta(q_i, >) = (q_j, >, E),$$

que serão simuladas por transições da forma:

$$\delta(q_i, B) = (q_j, B, E),$$

se $(<, q_0, \gamma >) \vdash^* (\lambda_1, q_f, \mu_1)$, $q_f \in F$, em M_1, então $(<, q_0, \gamma) \vdash^* (\lambda_2, q_f, \mu_2)$, $qf \in F$, em M_2. O símbolo B imediatamente à direita do último símbolo de γ em M_2 simula assim o símbolo de fim de cadeia "$>$" em M_1.

Logo, M_1 e M_2 aceitam a mesma linguagem e portanto L, além de ser uma linguagem sensível ao contexto, é também uma linguagem recursiva. ■

Teorema 6.3 (Sensíveis ao contexto $\neq$ recursivas) *A classe das linguagens sensíveis ao contexto constitui subconjunto próprio da classe das linguagens recursivas.*

Justificativa Basta demonstrar que existe pelo menos uma linguagem recursiva que não é sensível ao contexto. Como instância desse fato, podem-se citar todas as linguagens que incluem a cadeia vazia, uma vez que, conforme foi discutido anteriormente na Seção 5.1, as linguagens sensíveis ao contexto não incluem tais cadeias.

Um exemplo não-trivial, desta vez sem recorrer à cadeia vazia, é a linguagem L_R, apresentada no final da Seção 5.7, quando foi empregada para demonstrar a existência de linguagens que não são sensíveis ao contexto (Teorema 5.8).

Partindo-se (Teorema 5.7) de uma enumeração de todas as gramáticas sensíveis ao contexto $(G_1, G_2, ...)$, e também de uma enumeração de todas as cadeias possíveis de serem geradas sobre o alfabeto $\{a, b\}$ $(\alpha_1, \alpha_2, ...)$, o Teorema 5.8 mostra que a linguagem

$$L_R = \{\alpha_i \mid \alpha_i \notin L(G_i), \forall\, i \geq 1\}$$

não pode ser sensível ao contexto.

Por outro lado, qualquer que seja a cadeia $\beta \in \Sigma^*$, será sempre possível determinar mecanicamente se β pertence ou não a L_R, o que pode ser feito conforme o método descrito no Algoritmo 6.3.

Algoritmo 6.3 ($\beta \in L_R$?) *Determinação da pertinência de $\beta \in \Sigma^*$ a L_R, definida no Teorema 5.8.*

- Entrada: uma cadeia $\beta \in \Sigma^*$;
- Saída: SIM, se $\beta \in L_R$; NÃO, se $\beta \notin L_R$;
- Método:

    1. Compara-se β com cada uma das cadeias α_i, até que haja coincidência entre ambas;
    2. Seleciona-se a gramática G_i correspondente à cadeia α_i (lembrar que foi estabelecida uma função bijetora entre o conjunto das gramáticas e o conjunto das cadeias);
    3. Determina-se se $\alpha_i \in L(G_i)$. Como se trata de linguagens sensíveis ao contexto, tal determinação pode sempre ser efetuada;
    4. Se o resultado for que $\alpha_i \in L(G_i)$, então, por construção, $\alpha_i \notin L_R$ e a resposta é NÃO. Caso o resultado seja que $\alpha_i \notin L(G_i)$, então, por construção, $\alpha_i \in L_R$ e a resposta é SIM.

Logo, conforme o Algoritmo 6.3, é sempre possível determinar se uma cadeia β qualquer pertence a L_R. Em outras palavras, L_R é uma linguagem decidível e, portanto, recursiva, sem ser sensível ao contexto. ■

6.5 Linguagens que não são Recursivas

A classe das linguagens recursivas não é a mais abrangente que se conhece. Ao contrário, é possível demonstrar que existe pelo menos uma linguagem não-recursiva.

Teorema 6.4 (Linguagens não-recursivas) *Existem linguagens que não são recursivas.*

Justificativa É suficiente provar que existe pelo menos uma linguagem não-recursiva. Essa demonstração consiste na definição da linguagem L_U, conforme abaixo, e na prova, por contradição, de que a mesma não pode pertencer à classe das linguagens recursivas. Em outras palavras, que esta linguagem não é decidível.

$$L_U = \{C(M)w \in \Sigma^* \mid w \in L(M)\}$$

onde:

- $C(M)$ é a codificação de uma Máquina de Turing com fita limitada, cujo alfabeto de entrada é Σ, como cadeia sobre o próprio alfabeto Σ;
- $w \in \Sigma^*$ é uma cadeia de entrada qualquer para M.

A linguagem L_U é também conhecida como **Linguagem Universal**, uma vez que o problema de se determinar se uma certa cadeia $w \in L(M)$, para uma certa Máquina de Turing M, pode ser reduzido ao problema de se determinar se $C(M)w \in L_U$. Conforme será discutido no Capítulo 7, é possível provar que L_U, apesar de não ser recursiva, é uma linguagem recursivamente enumerável. Logo, existe uma Máquina de Turing que aceita L_U. Tal máquina, denotada M_U, é denominada **Máquina de Turing Universal**. Em outras palavras, $L_U = L(M_U)$.

L_U presume uma certa forma de codificação das Máquinas de Turing com fita limitada, codificação esta que, em princípio, pode ser feita sobre o mesmo alfabeto de entrada Σ de M. Em outras palavras: se M é uma Máquina de Turing sobre um alfabeto Σ, codifica-se M — denotada $C(M)$ — como uma cadeia sobre o próprio alfabeto Σ.

Essa codificação pode ser feita de várias maneiras. A forma escolhida é irrelevante, uma vez que os resultados obtidos dela independem.

L_U deve ser entendida, portanto, como a linguagem formada pelo conjunto das cadeias $C(M)w$, sendo M uma Máquina de Turing com fita limitada qualquer, com alfabeto de entrada Σ, e w uma cadeia qualquer sobre Σ^*, desde que w pertença à linguagem definida por M. Esta linguagem, adiante demonstrada como sendo não-recursiva, serve como base para a apresentação de um problema fundamental da computação, conhecido como **Problema da Parada** da Máquina de Turing.

A forma de codificação $C(M)$ de uma máquina M é, como foi antecipado, irrelevante. É apenas necessário estabelecer uma convenção que permita a correspondência unívoca entre cada máquina distinta e a cadeia correspondente que a representa, e também garantir que $C(M)$ e w sejam cadeias construídas sobre um mesmo alfabeto. Serão consideradas duas possibilidades de codificação:

1. Codificar a máquina M como uma cadeia sobre o próprio alfabeto de entrada Σ de M, deixando a cadeia de entrada w inalterada, resultando na cadeia combinada $C(M)w$;
2. Codificar não apenas M, mas também a própria cadeia de entrada w, como cadeias sobre um segundo alfabeto fixo Δ, resultando em cadeias do tipo $C(M)C(w)$.

Para efeito prático de demonstração de uma forma de codificação que possa ser aplicada a qualquer Máquina de Turing, com qualquer alfabeto de entrada, será adotada a segunda alternativa.

Apresenta-se a seguir, portanto, um esquema genérico de codificação de máquinas e cadeias de entrada quaisquer sobre o alfabeto $\Delta = \{a, b, c\}$, escolhido arbitrariamente. Note-se, na codificação proposta, que, apesar de o alfabeto Δ possuir apenas três símbolos, ela permite a representação de Máquinas de Turing com qualquer quantidade de símbolos em seu alfabeto de entrada Σ, e também da própria cadeia $w \in \Sigma^*$ a ser processada por M.

- Cada estado não-final do conjunto de estados $Q = \{q_0, q_1, q_2...q_n\}$ será representado, respectivamente, pela cadeia $aa, aaaa, aaaaaa...a^{2(n+1)}$ (ou seja, uma quantidade par de símbolos a);

  $C(q_i) = a^{2(i+1)}$, para $q_i \in Q - Q_F$;

- Cada estado final do conjunto de estados $Q = \{q_0, q_1, q_2...q_n\}$ será representado, respectivamente, pela cadeia $a, aaa, aaaaa...a^{2n+1}$ (ou seja, uma quantidade ímpar de símbolos a);

  $C(q_i) = a^{2i+1}$, para $q_i \in Q_F$;

- O símbolo especial "$<$" será representado pela cadeia ba;

  $C(B) = ba$;

- O símbolo especial "B" será representado pela cadeia bba;

  $C(B) = bba$;

- Cada elemento σ_n do alfabeto $\Sigma = \{\sigma_0, \sigma_1, \sigma_2...\sigma_n\}$ será representado, respectivamente, pela cadeia $bbba, bbbba, bbbbba...b^{n+3}a$;

  $C(\sigma_i) = b^{i+3}a,\ \sigma_i \in \Sigma$;

- O sentido de movimentação do cursor de acesso será representado como c (esquerda) ou cc (direita).

  $C(E) = c$;

  $C(D) = cc$;

Assim, cada uma das transições $\delta(q_i, \sigma_m) = (q_j, \sigma_n, E)$ de M poderá ser representada como:

- Se $q_i \notin F, q_j \notin F$, então $a^{2(i+1)}b^{m+3}aa^{2(j+1)}b^{n+3}ac$
- Se $q_i \in F, q_j \notin F$, então $a^{2i+1}b^{m+3}aa^{2(j+1)}b^{n+3}ac$
- Se $q_i \notin F, q_j \in F$, então $a^{2(i+1)}b^{m+3}aa^{2j+1}b^{n+3}ac$
- Se $q_i \in F, q_j \in F$, então $a^{2i+1}b^{m+3}aa^{2j+1}b^{n+3}ac$

De maneira análoga, caso o sentido de movimentação do cursor de acesso seja D e não E, basta substituir a subcadeia c por cc nas formas gerais acima. Por exemplo:

$$\delta(q_i, \sigma_m) = (q_j, \sigma_n, D) \quad \text{e} \quad a^{2(i+1)} b^{m+3} a a^{2(j+1)} b^{n+3} acc, \quad \text{com } q_i \notin F, q_j \notin F$$

A representação de $M = (Q, \Sigma, \Gamma, \delta, q_0, <, B, F)$ será efetivada da seguinte forma:

- Q e Σ podem ser inferidos a partir de δ;
- Se $\Sigma \neq \Gamma$, então codifica-se $\Sigma \cup \Gamma$;
- A função de transição δ será codificada como uma seqüência de cadeias concatenadas conforme a convenção apresentada individualmente para a codificação de cada transição (acima);
- q_0 será convencionado como sendo o primeiro estado a ser referenciado na definição de δ;
- "$<$" e B possuem codificações fixas sobre $\{a, b, c\}$;
- F poderá ser inferido a partir dos estados presentes na definição da função de transição que estejam codificados com uma quantidade ímpar de símbolos a.

Codificações como essa são úteis para a demonstração deste e de diversos outros teoremas, uma vez que permitem que Máquinas de Turing sejam representadas como cadeias de entrada (ou parte de cadeias de entrada) para outras Máquinas de Turing.

Exemplo 6.4 Seja a máquina M definida na Figura 6.5:

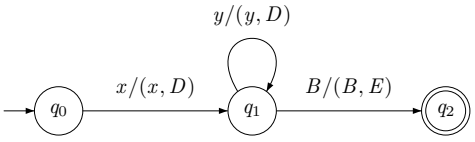

Figura 6.5: Máquina de Turing M que aceita xy^*

A codificação de seus componentes, de acordo com as convenções anteriormente apresentadas, ocorre conforme a Figura 6.6.

Q	q_0	aa
	q_1	$aaaa$
	q_2	$aaaaa$
-	$<$	ba
-	B	bba
Σ	x	$bbba$
	y	$bbbba$
δ	$\delta(q_0,x) = (q_1,x,D)$	$aabbbaaaaabbbacc$
	$\delta(q_1,y) = (q_1,y,D)$	$aaaabbbbaaaaabbbbacc$
	$\delta(q_1,B) = (q_2,B,E)$	$aaaabbaaaaaabbac$

Figura 6.6: Codificação dos elementos de M sobre $\{a, b, c\}$

A representação completa desta máquina é obtida concatenando-se as cadeias que representam as suas transições, conforme mostrado na Figura 6.7:

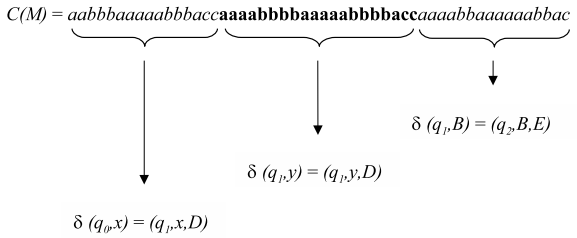

Figura 6.7: $C(M) : M$ como uma cadeia sobre $\{a, b, c\}$

□

A Figura 6.8 ilustra a codificação $C(w)$ para a cadeia de entrada $w = xyyB$.

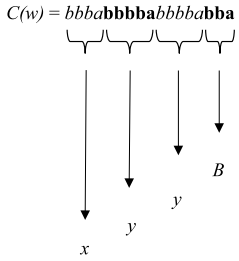

Figura 6.8: $C(w) : w$ como uma cadeia sobre $\{a, b, c\}$

A cadeia da Figura 6.9 representa a codificação de M seguida da codificação de uma cadeia w, ambas construídas sobre o mesmo alfabeto de codificação $\Delta = \{a, b, c\}$, ou seja, $C(M) - C(w)$. O símbolo especial "$-$" é usado para separar a codificação da máquina da codificação da cadeia:

C(M)-C(w) = aabbbaaaaabbbacc**aaaabbbbaaaaabbbbacc**aaaabbaaaaaabbac-bbba**bbbba**bbbba**bba**

Figura 6.9: $C(M) - C(w) : M$ e w como cadeias sobre $\{a, b, c\}$

Retorna-se agora ao Problema da Parada e à demonstração de que L_U não é recursiva. Para isso, será adotada a primeira forma de codificação sugerida, ou seja, aquela em que o alfabeto de codificação empregado é o próprio alfabeto de entrada da máquina M considerada. A segunda forma, exemplificada acima, também pode ser usada, sem que isso implique qualquer alteração nos resultados obtidos.

Suponha-se, portanto, que L_U seja uma linguagem recursiva, e portanto decidível. Isso acarreta a existência de uma Máquina de Turing com fita limitada M que decide L_U. Seja $C(N)w$ a cadeia formada pela concatenação da codificação da máquina N (sobre o alfabeto Σ) com a cadeia de entrada $w \in \Sigma^*$. A análise de $C(N)w$ por M é ilustrada na Figura 6.10.

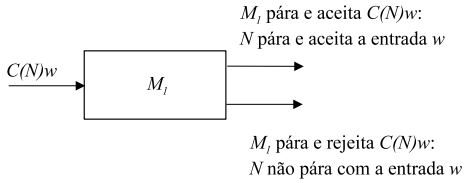

Figura 6.10: Problema da Parada: passo 1

Em seguida, constrói-se M_2 a partir de M_1, de modo que, para toda cadeia capaz de fazer M_1 parar, aceitando a entrada, M_2 deverá passar a executar uma seqüência infinita de movimentações. Caso M_1 pare, rejeitando a entrada, M_2 deverá também parar, rejeitando a entrada (figura 6.11).

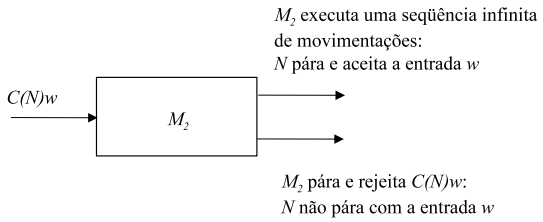

Figura 6.11: Problema da Parada: passo 2

Tal modificação é de fácil realização: basta fazer com que M_1, ao atingir uma configuração de aceitação, transite para um novo estado, criado especialmente para essa finalidade, em M_2, e lá permaneça indefinidamente lendo símbolos na fita de entrada e deslocando o cursor de acesso para a direita.

Seja M_3 uma Máquina de Turing com fita limitada que duplica a cadeia fornecida como entrada. Se a cadeia sobre a fita de entrada é γ, ao término do processamento a fita de trabalho conterá $\gamma\gamma$, conforme a Figura 6.12:

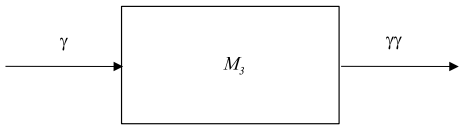

Figura 6.12: Problema da Parada: passo 3

Considere-se agora a máquina M_4, obtida pela combinação das máquinas M_3 e M_2, de tal forma que a saída de M_3 seja usada como entrada para M_2 (Figuras 6.13 e 6.14).

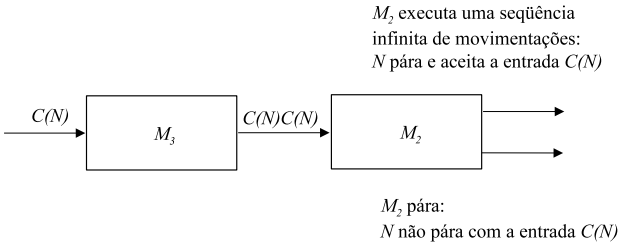

Figura 6.13: Problema da Parada: passo 4

ou ainda:

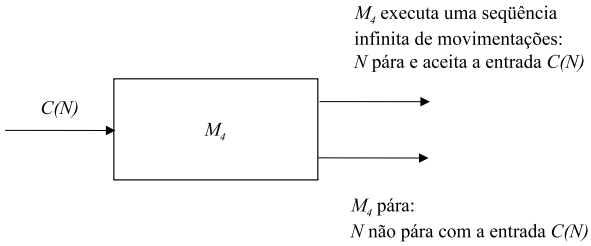

Figura 6.14: Problema da Parada: passo 5

Faça-se $N = M_4$, ou seja, submeta-se a M_4 uma codificação da própria máquina M_4. Isso é possível, uma vez que M_4 aceita cadeias sobre um certo alfabeto Σ, e $C(M_4)$ será codificada também como uma cadeia sobre este mesmo Σ. No caso considerado, $\Sigma = \{a, b, c\}$, porém qualquer outro alfabeto produziria os mesmos resultados (ver Figura 6.15).

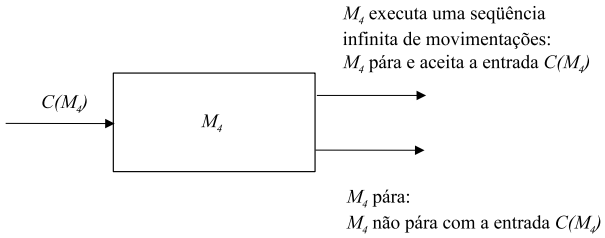

Figura 6.15: Problema da Parada: passo 6

A Figura 6.15 remete a uma contradição: por um lado, temos a informação de que, ao analisar a cadeia $C(M_4)$, se a máquina M_4 parar, então M_4 executa uma seqüência infinita de movimentações. Por outro, que ao analisar a cadeia $C(M_4)$, se M_4 não parar, então M_4 pára. Tem-se, portanto, uma contradição. Logo, a nossa hipótese inicial não é válida, ou seja, L_U não pode ser uma linguagem recursiva.

O Problema da Parada, além de demonstrar que a linguagem L_U não é recursiva, remete a uma outra conclusão importante: não existe solução para se determinar se uma certa máquina (ou programa) pára ao processar uma dada entrada. Naturalmente, este resultado é válido apenas para o caso de máquinas e entradas arbitrárias, não conhecidas *a priori*. Eventualmente, este problema pode ser resolvido para combinações de máquinas e/ou entradas pré-determinadas. ■

A linguagem L_K a seguir apresentada é também um exemplo clássico de linguagem não-recursiva:

$$L_K = \{ w_i \mid w_i \in L(M_i) \}$$

Esta linguagem pressupõe uma certa ordenação (por exemplo, lexicográfica) das cadeias w_i sobre um certo alfabeto Σ, e também do conjunto das Máquinas de Turing sobre o mesmo alfabeto. Ela compreende as cadeias w_i que são aceitas pelas Máquinas de Turing M_i correspondentes, e pressupõe uma bijeção entre o conjunto das máquinas M_i e o conjunto das cadeias $w_i, i \geq 0$, conforme a Tabela 6.1.

w_0	w_1	w_2	...	w_n	...
$\updownarrow$	$\updownarrow$	$\updownarrow$		$\updownarrow$	
M_0	M_1	M_2	...	M_n	...

Tabela 6.1: Bijeção entre cadeias e Máquinas de Turing

Teorema 6.5 (L_K não-recursiva) *A linguagem L_K não é recursiva.*

Justificativa Conforme o Teorema 7.19 (apresentado mais adiante), uma linguagem L é recursiva se e somente L e seu complemento forem recursivamente enumeráveis. O Teorema 7.13 (também apresentado mais adiante), por sua vez, prova que a linguagem $L_D = \Sigma^* - L_K$ (o complemento de L_K) não é recursivamente enumerável. Logo, L_K não é recursiva. ■

6.6 Propriedades de Fechamento

A seguir serão demonstradas algumas das propriedades de fechamento mais importantes das linguagens recursivas. Esta classe de linguagens é fechada em relação às operações de:

- União
- Concatenação
- Complementação
- Intersecção

As respectivas demonstrações serão apresentadas na seqüência. As linguagens recursivas, no entanto, não são fechadas em relação à operação de:

- Fechamento reflexivo e transitivo (Fecho de Kleene)

resultado este que não será demonstrado neste texto, podendo ser encontrado em [46].

Teorema 6.6 (Fecho na união) *A classe das linguagens recursivas é fechada em relação à operação de união.*

Justificativa Sejam L_1 e L_2 duas linguagens recursivas quaisquer. Então, L_1 é decidida por uma Máquina de Turing M_1 e L_2 é decidida por uma Máquina de Turing M_2. A linguagem $L_3 = L_1 \cup L_2$ é decidida por M_3, construída de acordo com o Algoritmo 6.4.

Algoritmo 6.4 (Fecho na união) *Obtenção de uma Máquina de Turing que decide a união de duas linguagens, a partir das Máquinas de Turing que decidem cada uma das linguagens.*

- Entrada: M_1 e M_2, duas Máquinas de Turing que decidem, respectivamente, as linguagens L_1 e L_2;
- Saída: uma Máquina de Turing M_3 que decide a linguagem $L_1 \cup L_2$;
- Método:

    1. M_3 inicialmente simula M_1 com uma cadeia de entrada w;
    2. Se M_1 aceita w, então M_3 pára e aceita w;
    3. Se M_1 rejeita w, então M_3 simula M_2 com a mesma cadeia w;
    4. Se M_2 aceita w, então M_3 pára e aceita w;
    5. Se M_2 rejeita w, então M_3 pára e rejeita w;

Conforme o Algoritmo 6.4, M_3 aceita w se e somente se $w \in L_1$ ou $w \in L_2$. Caso contrário, M_3 rejeita w. Logo, L_3 é uma linguagem recursiva. ■

Teorema 6.7 (Fecho na concatenação) *A classe das linguagens recursivas é fechada em relação à operação de concatenação.*

Justificativa Sejam L_1 e L_2 duas linguagens recursivas quaisquer. Então, L_1 é decidida por uma Máquina de Turing M_1 e L_2 é decidida por uma Máquina de Turing M_2. A linguagem $L_3 = L_1 L_2$ é decidida por M_3, construída de acordo com o Algoritmo 6.5.

Algoritmo 6.5 (Fecho na concatenação) *Obtenção de uma Máquina de Turing que decide a concatenação de duas linguagens, a partir das Máquinas de Turing que decidem cada uma das linguagens.*

- Entrada: M_1 e M_2, duas Máquinas de Turing que decidem, respectivamente, as linguagens L_1 e L_2;
- Saída: uma Máquina de Turing M_3 que decide a linguagem $L_1 L_2$;
- Método:

    1. M_3 inicialmente simula M_1 com uma cadeia de entrada $w, w = \alpha\beta$;

2. Se M_1 aceita uma subcadeia α de w, então M_3 simula M_2 com a subcadeia β;

3. Se M_2 rejeita β, então M_3 pára e rejeita a cadeia w;

4. Se M_1 não aceita nenhuma subcadeia α de w, então M_3 pára e rejeita w.

Conforme o Algoritmo 6.5, M_3 aceita w se e somente se $\alpha \in L_1$ e $\beta \in L_2$. Caso contrário, M_3 rejeita w. Logo, L_3 é uma linguagem recursiva. ■

Teorema 6.8 (Fecho na complementação) *A classe das linguagens recursivas é fechada em relação à operação de complementação.*

Justificativa Seja L_1 uma linguagem recursiva qualquer definida sobre um alfabeto Σ. Então, L_1 é aceita por uma Máquina de Turing M_1 que sempre pára, qualquer que seja a entrada. A linguagem $L_2 = \Sigma^* - L_1$ é decidida por M_2 construída de acordo com o Algoritmo 6.6.

Algoritmo 6.6 (Fecho na complementação) *Obtenção de uma Máquina de Turing que decide o complemento de uma linguagem, a partir da Máquina de Turing que decide a linguagem.*

- Entrada: M_1, uma Máquina de Turing que decide a linguagem L_1;
- Saída: uma Máquina de Turing M_2 que decide a linguagem $\Sigma^* - L_1$;
- Método:

  1. M_2 simula M_1 com uma cadeia de entrada w;
  2. Se M_1 aceita w, então M_2 pára e rejeita w;
  3. Se M_1 rejeita w, então M_1 pára e aceita w.

Conforme o Algoritmo 6.6, M_2 aceita w se e somente se $w \in \Sigma^* - L_1$. Caso contrário, ou seja, se $w \in L_1$, então M_2 rejeita w. Logo, L_2 é uma linguagem recursiva. ■

Teorema 6.9 (Fecho na intersecção) *A classe das linguagens recursivas é fechada em relação à operação de intersecção.*

Justificativa Conforme demonstrado no Teorema 3.18, o fechamento de uma classe de linguagens em relação à operação de intersecção pode ser verificado a partir do fechamento em relação às operações de união e complementação. Como essas propriedades já foram demonstradas nos Teoremas 6.6 e 6.8, fica demonstrado o fechamento da classe das linguagens recursivas em relação à operação de intersecção. ■

6.7 Questões Decidíveis e Não-Decidíveis

A classe das linguagens recursivas não oferece, infelizmente, resposta para a maioria das questões mais comuns que se costuma formular a seu respeito. Tais questões encontram-se relacionadas abaixo.

Sejam L_1 e L_2 duas linguagens recursivas quaisquer. Então:

- $L_1 = \varnothing$?
- $L_1 = \Sigma^*$?
- $L_1 = L_2$?
- $L_1 \subseteq L_2$?
- $L_1 \cap L_2 = \varnothing$?

É possível demonstrar que as linguagens formadas por essas questões não são recursivas. Logo, nem sempre será possível determinar uma resposta, qualquer que seja ela, para questões dessa natureza envolvendo as próprias linguagens recursivas. As demonstrações não serão feitas neste texto, porém podem ser encontradas em [46].

Por outro lado, conforme mostra o Teorema 6.10, é sempre possível decidir a questão "$w \in L$?" para a classe das linguagens recursivas.

Teorema 6.10 (A cadeia pertence?) *Dadas uma cadeia $w \in \Sigma^*$ e uma linguagem recursiva L, $L \subseteq \Sigma^*$, é sempre possível determinar se $w \in L$.*

Justificativa Conforme a própria definição de linguagem recursiva, se L é uma linguagem dessa classe, então existe uma Máquina de Turing M que reconhece (decide) L. Para determinar se uma certa cadeia $w \in L(M)$, basta submeter w à análise de M. Após um número finito de movimentações, M irá parar indicando se w pertence ou não a L. ■

7 Linguagens Recursivamente Enumeráveis

Uma linguagem L é dita **recursivamente enumerável** (ou simplesmente **irrestrita**) se for aceita por pelo menos uma Máquina de Turing M. Ou seja:

1. Para toda cadeia $w \in L, M$ pára e aceita w;

2. Para toda cadeia $z \in \Sigma^* - L, M$ pára e rejeita z ou executa uma seqüência infinita de movimentações.

Uma linguagem L é dita **estritamente recursivamente enumerável** se, para toda e qualquer Máquina de Turing M que aceita L, existir pelo menos uma cadeia $z \in \Sigma^* - L$, tal que M inicie uma seqüência interminável de movimentações em seu processamento.

Da mesma forma que as linguagens recursivas, e diferentemente das linguagens regulares, livres de contexto e sensíveis ao contexto, não se conhece qualquer tipo de estrutura que permita identificar as linguagens recursivamente enumeráveis apenas pela inspeção das propriedades sintáticas de suas sentenças.

As linguagens recursivamente enumeráveis são, portanto, inicialmente caracterizadas a partir das Máquinas de Turing. Conforme será estudado adiante, elas também podem ser caracterizadas por meio do modelo mais geral de gramáticas, as chamadas gramáticas irrestritas (ver Seção 7.3).

A maior importância desta classe de linguagens reside em sua aplicação ao desenvolvimento teórico da computação.

Conforme estudado no Capítulo 6, as linguagens recursivas são também conhecidas como linguagens decidíveis. Isso ocorre porque, para tal classe de linguagens, é sempre possível obter uma Máquina de Turing que sempre pára em resposta a qualquer entrada que lhe seja submetida, não importa se aceitando ou rejeitando tal entrada. Cadeias que pertencem à linguagem são aceitas e cadeias que não lhe pertencem são rejeitadas. Essa característica é particularmente útil quando se deseja estudar a computabilidade de determinados tipos de problema, em particular dos chamados problemas de decisão.

7.1 Decidibilidade

Um **problema de decisão** é um problema cuja formulação conduz a apenas duas respostas possíveis: SIM ou NÃO. Exemplo: dadas duas linguagens regulares L_1 e L_2 quaisquer, é sempre possível determinar se $L_1 = L_2$? Para cada par de linguagens considerado, este problema admite apenas uma resposta entre duas possíveis.

Uma **instância de um problema** é um caso particular de um problema geral, com seus argumentos completamente definidos. Exemplo: dadas a linguagem regular L_1 (segue a especificação de L_1) e a linguagem regular L_2 (segue a especificação de L_2), será que $L_1 = L_2$?

Se todas as instâncias de um certo problema, para as quais as respostas forem afirmativas (SIM), forem codificadas sobre um alfabeto Σ qualquer, tal conjunto de cadeias forma uma linguagem L sobre Σ. Note-se que as instâncias de problemas cuja resposta é negativa não pertencem a L.

Se a linguagem assim constituída for recursiva, é sabido que existe pelo menos uma Máquina de Turing que decide L. Ou seja, dada uma instância qualquer do problema, cuja resposta seja desconhecida, será sempre possível determinar se a resposta é afirmativa ou negativa, bastando para isso verificar se a referida Máquina de Turing aceita ou rejeita a cadeia que representa a instância.

A importância deste resultado está no fato de que linguagens que representam problemas gerais, uma vez identificadas como sendo recursivas, são tais que permitem a determinação mecânica da solução, qualquer que seja a instância considerada. A mecanização da solução, no caso, é implementada pela Máquina de Turing que aceita L.

As linguagens recursivamente enumeráveis, por outro lado, não gozam dessa propriedade. Conforme a sua definição, as cadeias não pertencentes à linguagem podem tanto ser rejeitadas após uma parada como provocar a execução de uma seqüência interminável de movimentações na Máquina de Turing correspondente.

Este fenômeno, característico e específico das linguagens estritamente recursivamente enumeráveis, não ocorre com as outras classes de linguagens anteriormente consideradas. De fato, é possível provar que, qualquer que seja a linguagem regular, livre de contexto ou recursiva em questão, é sempre possível obter um autômato finito, um autômato de pilha ou uma Máquina de Turing com fita limitada que sempre pára, qualquer que seja a cadeia de entrada. Isso não significa, no entanto, que não existam reconhecedores desses tipos que eventualmente executem seqüências infinitas de movimentações em resposta a alguma cadeia de entrada.

Considere-se agora um outro problema, cujas instâncias de resposta afirmativa também possam ser codificadas como cadeias sobre um certo alfabeto de entrada. Considere-se ainda que a linguagem formada por essa coleção de cadeias seja do tipo recursivamente enumerável. Isso acarreta a impossibilidade de se verificar, mecanicamente, se determinada cadeia pertence ou não à linguagem, pois cadeias não pertencentes à linguagem eventualmente poderão provocar uma movimentação interminável da Máquina de Turing correspondente.

Logo, torna-se impossível, no caso geral, determinar se um certo problema possui solução, qualquer que seja a instância considerada. Se determinada instância possui resposta SIM (caso em que a cadeia que a representa pertence à linguagem), é fato que a correspondente Máquina de Turing irá parar após um tempo finito de processamento. Se a resposta é NÃO, sabe-se apenas que o processamento poderá eventualmente parar, rejeitando a entrada, ou então iniciar uma seqüência infindável de movimentações.

Para um observador externo, não há nenhuma garantia, para qualquer que seja a instância que venha a ser eventualmente considerada, de que o processamento irá parar em algum momento, produzindo algum resultado.

Devido a essa característica, as linguagens recursivamente enumeráveis são também denominadas **indecidíveis**. Problemas cujas representações na forma de linguagens sejam recursivamente enumeráveis são também chamados de problemas indecidíveis.

De uma forma geral, os termos "recursivo" e "recursivamente enumerável" são empregados quando se trata de linguagens genéricas, e os termos "decidível" (sinônimo de "recursiva") e "indecidível" ou "não-decidível" (sinônimos de "recursivamente enumerável"), quando se trata de linguagens que representam problemas.

O termo **decidibilidade** refere-se ao estudo das linguagens formais, com vistas à determinação das classes a que estas pertencem.

Os termos **solucionável**, **não-solucionável** (ou **insolúvel**) e **parcialmente solucionável**, também empregados quando se trata de problemas, significam, respectivamente, que as correspondentes linguagens são: (i) recursivas, ou seja, podem sempre ser decididas no caso geral; (ii) recursivamente enumeráveis, ou seja, não podem ser decididas no caso geral; e (iii) recursivamente enumeráveis, enfatizando o fato de que pode haver solução para algumas instâncias do problema (ainda que correndo o risco de se esperar indefinidamente por uma resposta).

Aplicada ao estudo dos problemas de decisão, a decidibilidade indica se os mesmos são solucionáveis, não-solucionáveis ou parcialmente solucionáveis.

É fácil, portanto, justificar o grande interesse prático que existe por problemas para os quais se possa provar que a linguagem de representação é recursiva. Problemas cuja linguagem de representação seja comprovadamente estritamente recursivamente enumerável, por outro lado, servem sobretudo para demonstrar a inexistência de procedimentos mecânicos (algoritmos) que possam resolvê-los no caso geral. E isso é um resultado teórico excepcional, uma vez que evita o desperdício de recursos na busca de soluções gerais e mostra que determinados problemas não possuem solução, independentemente da tecnologia computacional — atual ou futura — que possa ser empregada na busca de pretensas soluções.

Note-se, finalmente, que o fato de um determinado problema poder ser resolvido no caso geral não significa que os algoritmos de resolução conhecidos sejam necessariamente eficientes. Em muitas situações, a busca de solução para o caso geral, cuja existência é conhecida através da teoria, pode ser inviabilizada na prática, em virtude do imenso custo associado à sua realização (custo esse refletido, usualmente, no volume de memória necessário ou no tempo de processamento requerido para se chegar às soluções).

Por outro lado, a inexistência de solução para um problema no caso geral não significa que ele não possa ser resolvido para instâncias específicas ou pré-determinadas, ou, ainda, para conjuntos de instâncias pré-determinadas do problema original.

7.2 Máquinas de Turing como Enumeradoras de Linguagens

Originalmente, o termo "recursivamente enumerável" refere-se ao fato de as sentenças de uma linguagem deste tipo poderem ser enumeradas (listadas ou contadas) através de um procedimento mecanizado — um algoritmo ou uma Máquina de Turing.

Diz-se que a Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, <, B, q_0, F)$ **enumera uma linguagem**, quando:

- M possui pelo menos duas fitas de entrada;
- Uma das fitas é usada apenas para a gravação de cadeias geradas sobre o alfabeto Σ. Nessa fita, o cursor de acesso desloca-se apenas para a direita, símbolos podem apenas ser gravados, e cadeias consecutivas são separadas pelo símbolo "#", não pertencente a Σ.

O conjunto das cadeias gravadas por M na fita de saída é denotado por $E(M)$, e representa a linguagem enumerada por M. $E(M) \subseteq \Sigma^*$ pode ser finita ou infinita. Nesse último caso, o processamento de M não pára nunca e M está sempre gravando novas cadeias na fita de saída, uma após a outra.

Os Teoremas 7.1 e 7.2, a seguir apresentados, mostram que uma linguagem L é recursivamente enumerável se e somente se ela puder ser enumerada por uma Máquina de Turing M, ou seja, $L = E(M)$.

Teorema 7.1 (Enumeradas ⇒ recursivamente enumeráveis) *Seja $L = E(M)$. Então L é recursivamente enumerável.*

Justificativa Para demonstrar que L é recursivamente enumerável, é suficiente demonstrar que existe uma Máquina de Turing N que aceita L. O Algoritmo 7.1 a seguir mostra como construir N a partir de M.

Algoritmo 7.1 (Enumeradas ⇒ recursivamente enumeráveis) *Obtenção de uma Máquina de Turing que aceita a linguagem gerada por uma Máquina de Turing que a enumera.*

- Entrada: uma Máquina de Turing M que enumera uma linguagem L;
- Saída: uma Máquina de Turing N que aceita L;
- Método:

    1. N possui duas fitas de entrada: na primeira serão gravadas as cadeias de L, uma a uma (através da simulação de M por N), e na segunda será gravada a cadeia w que se deseja verificar se pertence ou não a L;
    2. N opera como M: gerando uma nova cadeia de L e gravando-a na fita de saída;
    3. Antes, porém, de acrescentar o delimitador "#" à sua direita e proceder à geração de uma nova cadeia, N compara a cadeia w contida na segunda fita com a cadeia recém-gerada na primeira fita;
    4. Se forem iguais, N pára e aceita w. Caso contrário, desviar para (2).

De acordo com o Algoritmo 7.1, é claro que, se $w \in L$, então w é aceita por N, pois w será inevitavelmente gerada e gravada por N na primeira fita, em um intervalo de tempo finito.

Se, por outro lado, $w \notin L$, N pode tanto parar rejeitando w (no caso em que L for finita), como entrar em um processamento interminável, dentro do qual novas cadeias são sucessivamente geradas e comparadas com w (no caso em que L é infinita). L é, portanto, uma linguagem recursivamente enumerável. ■

Teorema 7.2 (Enumeradas ⇐ recursivamente enumeráveis) *Seja L uma linguagem recursivamente enumerável. Então existe N, tal que $L = E(N)$.*

Justificativa Se L é recursivamente enumerável, então $L = L(M)$ para alguma Máquina de Turing M. Uma primeira tentativa de demonstração consiste em construir uma máquina N que simula M, conforme o Algoritmo 7.2.

Algoritmo 7.2 (Enumeradas ⇐ recursivamente enumeráveis, versão 1) *Obtenção de uma Máquina de Turing que enumera uma linguagem definida por outra Máquina de Turing.*

- Entrada: uma Máquina de Turing M;
- Saída: uma Máquina de Turing N que enumera $L(M)$;
- Método:

    1. N gera uma nova cadeia $w \in \Sigma^*$, onde Σ é o alfabeto de L (essas cadeias devem ser geradas em uma certa ordem e sucessivamente. Por exemplo, pode-se considerar a ordenação lexicográfica das cadeias de L na seqüência de comprimento crescente).
    2. A cadeia gerada w é escrita em uma fita auxiliar de N (fita 1).
    3. A fita 2 de N contém uma codificação $C(M)$ da máquina M.
    4. N simula M com a cadeia w gravada na fita 1.
    5. Caso M aceite w, $w\#$ é escrito na fita 3.
    6. Desviar para (1).

O procedimento do Algoritmo 7.2 consiste em gerar, de forma sistemática e exaustiva, todas as cadeias de Σ^*, testando cada uma delas em M para determinar se pertencem ou não a L. Em caso afirmativo, elas são individualmente copiadas para a fita de saída de N, que dessa forma enumera as cadeias de L, ou seja, $L = E(N)$.

Essa solução funcionaria perfeitamente, exceto por um detalhe: se L for uma linguagem estritamente recursivamente enumerável, isso significa que existe pelo menos uma cadeia $z \in \Sigma^* - L$, de modo que M inicia uma seqüência infindável de movimentações em sua operação. Logo, N inicia essa seqüência no passo (4) do procedimento, e as demais cadeias de L não são geradas, nem testadas ou gravadas por N em sua fita de saída.

Se, no entanto, L for uma linguagem decidível (recursiva), então o procedimento pode ser usado sem qualquer risco de induzir M, e conseqüentemente N, a um processamento que pare de gerar resultados.

Não obstante, é possível adotar um procedimento ligeiramente modificado que permita superar as limitações do procedimento analisado, servindo, indistintamente, tanto para linguagens recursivas quanto para linguagens recursivamente enumeráveis.

Seja L uma linguagem recursivamente enumerável, $L \subseteq \Sigma^*$ e $w \in L$. Então, se α_0, α_1, α_2, α_3, ..., α_n, ... é uma ordenação lexicográfica, de comprimento crescente, das cadeias de Σ^*, é fato que $w = \alpha_i$, para algum valor de i, e apenas um.

Exemplo 7.1 Considere $\Sigma = \{a, b\}$. A ordenação lexicográfica de comprimento crescente das cadeias de Σ^* é:

- Comprimento 0: ϵ
- Comprimento 1: a, b
- Comprimento 2: aa, ab, ba, bb
- Comprimento 3: $aaa, aab, aba, abb, baa, aba, bba, bbb$
- ...

- Comprimento n: $a^n, a^{n-1}b, ..., b^{n-1}a, b^n$
- ...

□

Além disso, como $w \in L$, também é fato que w é aceita por uma Máquina de Turing M após um número limitado de movimentações que levam M desde a sua configuração inicial até uma configuração final qualquer. Supondo que M seja determinística, e sendo $|w| = k$, w é aceita apenas na k-ésima configuração assumida por M.

Defina-se a função parcial $\gamma : \mathbb{Z}_+ \times \mathbb{Z}_+ \rightarrow \Sigma^*$ da seguinte forma: se $\alpha_i \in L, |\alpha_i| = j$ e α_i é aceita por uma Máquina de Turing M determinística (neste caso, após j configurações), então γ associa o par (i, j) com a cadeia α_i. A função γ é injetora e sobrejetora, pois não existem dois pares que possam ser associados com a mesma sentença, e todas as sentenças de L estão associadas com exatamente um par.

Considere-se um certo par (p, q) escolhido ao acaso. Se $\alpha_p \in L$, é certo que α_p é aceita por M após um certo número de movimentações. Se este número for q, a cadeia α_p será enumerada. Caso contrário, ela será descartada, até que o par correto seja sorteado (note-se que para cada valor de p existe uma quantidade infinita de pares com este valor no primeiro elemento da dupla: $...(p, q-2), (p, q-1), (p, q), (p, q+1), (p, q+2)...$ etc.).

O objetivo agora é o de construir uma Máquina de Turing N que, partindo de uma seqüência de todos os pares ordenados $(i, j), i \geq 1, j \geq 1$, em que cada (i, j) ocorre uma única vez, testará sistematicamente se a cadeia α_i é aceita por M em j movimentos. Em caso afirmativo, α_i será enumerada. Em caso afirmativo ou negativo, um novo par é considerado e um novo teste é efetuado.

Dessa forma, evita-se testar α_i em M de forma contínua, até que M pare (conforme a proposta do Algoritmo 7.2), pois isso poderia causar problemas no caso de linguagens estritamente recursivamente enumeráveis. Nessa nova técnica, o teste de cada α_i é efetuado apenas durante uma seqüência limitada de configurações (porém suficiente para a sua aceitação, caso a cadeia pertença à linguagem), evitando-se dessa maneira a ocorrência de eventuais seqüências intermináveis de movimentações. Assim, a sentença α_i é enumerada apenas quando o par (i, j) correspondente tiver sido considerado.

A fim de garantir a enumeração sistemática de todas as sentenças de L, é necessário considerar a geração e o teste sistemático de todos os pares (i, j), sem repetição, um de cada vez. Conforme estudado no Capítulo 1 (ver Exemplo 1.54), o conjunto desses pares pode facilmente ser enumerado considerando-se a organização em que os pares cuja soma representa os mesmos valores são agrupados em ordem crescente do valor da soma e, dentro de cada um desses grupos, na ordem descrescente do primeiro termo e crescente do segundo:

- Soma 2: $(1, 1)$
- Soma 3: $(2, 1), (1, 2)$
- Soma 4: $(3, 1), (2, 2), (1, 3)$
- Soma 5: $(4, 1), (3, 2), (2, 3), (1, 4)$
- Soma 6: $(5, 1), (4, 2), (3, 3), (2, 4), (1, 5)$
- ...
- Soma n: $(n-1, 1), (n-2, 2), (n-3, 3)...(3, n-3), (2, n-2), (1, n-1)$

- ...

Por outro lado, dado um valor de i, será necessário gerar a cadeia α_i correspondente. É possível, sem grande dificuldade, demonstrar a existência de Máquinas de Turing que realizam essas duas tarefas, e que serão embutidas em N. O Algoritmo 7.3 corresponde à versão revisada do Algoritmo 7.2.

Algoritmo 7.3 (Enumeradas $\Leftarrow$ recursivamente enumeráveis, versão 2) *Revisão do Algoritmo 7.2 que evita seqüências infinitas de movimentações, sendo portanto adequada para linguagens estritamente recursivamente enumeráveis.*

- Entrada: uma Máquina de Turing M;
- Saída: uma Máquina de Turing N que enumera $L(M)$;
- Método:

  1. N gera o próximo par (i, j) da seqüência;
  2. N gera a cadeia $\alpha_i \in \Sigma^*$;
  3. A cadeia gerada α_i é gravada em uma fita auxiliar de N (fita 1);
  4. A fita 2 de N contém uma codificação $C(M)$ da máquina M;
  5. N simula M com a cadeia α_i gravada na fita 1, efetuando j movimentações;
  6. Caso M aceite α_i na j-ésima configuração, então a cadeia $\alpha_i\#$ é gravada na fita 3, imediatamente após o final da cadeia anteriormente gravada nessa mesma fita;
  7. Desviar para (1).

Como o número de passos necessários para a geração de cada par (i, j) e de cada cadeia α_i é finito, o mesmo ocorrendo com o número de passos alocado para testar cada α_i em M, e como, além disso, todos os pares (i, j) são considerados (o que significa dizer que todas as cadeias de Σ^* são consideradas), pode-se afirmar que N enumera todas as sentenças de L, e apenas essas.

Se L for uma linguagem recursiva, e apenas neste caso, N enumera as sentenças de L em ordem lexicográfica, na seqüência de comprimento crescente. ■

7.3 Gramáticas Irrestritas

Uma gramática $G = (V, \Sigma, P, S)$ é dita **irrestrita** se nenhuma restrição adicional for aplicada às suas regras de produção, que devem seguir a forma geral $\alpha \rightarrow \beta, \alpha \in V^*NV^*, \beta \in V^*$. Observar que as gramáticas regulares, as livres de contexto e as sensíveis ao contexto são casos particulares das gramáticas irrestritas.

Note-se que, diferentemente do que ocorre com as gramáticas sensíveis ao contexto, as gramáticas irrestritas admitem que o lado direito das regras (β) possua um número de

símbolos menor que o lado esquerdo correspondente (α). Isso faz com que possa haver redução no tamanho das formas sentenciais durante o processo de derivação de sentenças.

Alguns autores consideram um formato ainda mais geral para as regras das gramáticas irrestritas: $\alpha \in V^+, \beta \in V^*$. Este formato permite que o lado esquerdo das regras seja formado por qualquer combinação de símbolos terminais e não-terminais, incluindo cadeias formadas exclusivamente por símbolos terminais, o que não é permitido no formato aqui adotado.

Na verdade, ambas as definições são equivalentes, podendo ser usadas indistintamente. A conversão de regras do formato mais geral ($\alpha \in V^+, \beta \in V^*$) para o formato menos geral ($\alpha \in V^*NV^*, \beta \in V^*$) pode ser feita aplicando-se a técnica utilizada na demonstração do Teorema 5.1, em que novos símbolos não-terminais são criados e substituem todas as ocorrências de símbolos terminais, e a gramática é acrescida de produções unitárias que convertem cada um desses novos símbolos não-terminais nos símbolos terminais correspondentes.

Exemplo 7.2 A gramática $G_1 = (\{S, A, B, C, a, b, c\}, \{a, b, c\}, P_1, S)$, com P_1:

$$\begin{array}{rcl} \{S & \rightarrow & aAbcC \\ bc & \rightarrow & B \\ ABC & \rightarrow & b\} \end{array}$$

pode ser facilmente convertida para $G_2 = (\{S, A, B, C, X, Y, Z, a, b, c\}, \{a, b, c\}, P_2, S)$, com P_2:

$$\begin{array}{rcl} \{S & \rightarrow & XAYZC \\ YZ & \rightarrow & B \\ ABC & \rightarrow & Y \\ X & \rightarrow & a \\ Y & \rightarrow & b \\ Z & \rightarrow & c\} \end{array}$$

□

As convenções e a terminologia adotadas na apresentação e discussão dos demais tipos de gramática também são aplicáveis ao caso das gramáticas irrestritas. Derivações, formas sentenciais e a linguagem gerada pela gramática são denotadas de forma idêntica àqueles casos.

Exemplo 7.3 A gramática $G = (\{S, A, C, a, b, c\}, \{a, b, c\}, P, S)$, com P:

$$\begin{array}{rcl} \{S & \rightarrow & aAbc \\ A & \rightarrow & aAbC \mid \epsilon \\ Cb & \rightarrow & bC \\ Cc & \rightarrow & cc\} \end{array}$$

gera a linguagem $a^i b^i c^i, i \geq 1$. Esta mesma linguagem foi anteriormente formulada no Capítulo 5 através de uma gramática sensível ao contexto (ver Exemplo 5.14). No presente exemplo, ela é formalizada através de uma gramática irrestrita (por causa da produção $A \rightarrow \epsilon$, em que $|A| > |\epsilon|$). Exemplos de derivação:

- $S \Rightarrow aAbc \Rightarrow abc$
- $S \Rightarrow aAbc \Rightarrow aaAbCbc \Rightarrow aaaAbCbCbc \Rightarrow aaabCbCbc \Rightarrow aaabCbbCc \Rightarrow aaabbCbCc \Rightarrow aaabbbCCc \Rightarrow aaabbbCcc \Rightarrow aaabbbccc$

A aplicação da produção $S \rightarrow aAbc$, seguida de n aplicações da produção $A \rightarrow aAbC$ e, finalmente, da aplicação da produção $A \rightarrow \epsilon$, gera uma forma sentencial do tipo $a^{n+1}(bC)^n bc$. A transposição dos símbolos "b" para o lado esquerdo, junto aos símbolos "a", é feita com o auxílio da produção $Cb \rightarrow bC$. Após a aplicação dessa produção $n+1$ vezes, chega-se à forma sentencial $a^{n+1}b^{n+1}C^n c$. Finalmente, a aplicação da produção $Cc \rightarrow cc$ por n vezes substitui os não-terminais "C" pelos terminais "c" correspondentes, conduzindo à geração da sentença $a^{n+1}b^{n+1}c^{n+1}$. □

Exemplo 7.4 A gramática $G = (\{S, B, X, a, b\}, \{a, b\}, P, S)$, com P:

$$\begin{array}{rcl} \{S & \rightarrow & aBSa \mid aBXa \\ Ba & \rightarrow & aB \\ BX & \rightarrow & Xb \\ aX & \rightarrow & a\} \end{array}$$

gera a linguagem $a^i b^i a^i, i \geq 1$. A produção $aX \rightarrow a$ caracteriza esta gramática como sendo irrestrita, uma vez que $\mid aX \mid > \mid a \mid$.

As duas produções iniciais geram formas sentenciais do tipo $(aB)^n Xa^n$. A partir desse ponto, a aplicação repetida da produção $Ba \rightarrow aB$ permite obter formas sentenciais do tipo $a^n B^n Xa^n$. Resta, portanto, substituir os símbolos não-terminais "B" por terminais "b" para gerar as sentenças pretendidas, tarefa esta que é cumprida pelas três últimas produções. O não-terminal "X" serve como delimitador, separando as cadeias à sua esquerda e à sua direita. Ele é usado, inicialmente, como referência para substituir os símbolos "B" por "b" e, finalmente, para se auto-remover da forma sentencial, gerando $a^n b^n a^n$. Exemplos de derivação:

- $S \Rightarrow aBXa \Rightarrow aXba \Rightarrow aba$
- $S \Rightarrow aBSa \Rightarrow aBaBSaa \Rightarrow aBaBaBXaaa \Rightarrow aaBBaBXaaa \Rightarrow aaBaBBXaaa \Rightarrow aaaBBBXaaa \Rightarrow aaaBBXbaaa \Rightarrow aaaBXbbaaa \Rightarrow aaaXbbbaaa \Rightarrow aaabbbaaa$

□

7.4 Forma Normal para Gramáticas Irrestritas

Devido ao fato de as gramáticas regulares, livres de contexto e sensíveis ao contexto constituírem casos particulares das gramáticas irrestritas, a forma normal que será apresentada a seguir pode ser aplicada indistintamente em qualquer tipo de gramática examinada até o momento.

A demonstração do teorema seguinte baseia-se na demonstração de [55], que por sua vez é uma generalização da forma conhecida como Forma Normal de Kuroda para gramáticas sensíveis ao contexto (ver Seção 5.3).

Teorema 7.3 (Forma normal para gramáticas irrestritas) *Toda gramática irrestrita $G_1 = (V_1, \Sigma, P_1, S)$ define uma linguagem L que também pode ser gerada por uma outra gramática $G_2 = (V_2, \Sigma, P_2, S)$, equivalente, cujas produções são todas das seguintes formas: (1) $S \rightarrow \epsilon$; (2) $A \rightarrow \sigma$; (3) $A \rightarrow B$; (4) $A \rightarrow BC$; (5) $AB \rightarrow AC$; (6) $AB \rightarrow CB$; (7) $AB \rightarrow C$, onde $S, A, B, C \in N_2$ e $\sigma \in \Sigma$.*

Justificativa Conforme o Algoritmo 7.4, descrito a seguir, o qual incorpora as seguintes etapas:

i. Eliminação das produções vazias;

ii. Incorporação da produção (1), caso a cadeia vazia faça parte da linguagem;

iii. Substituição dos símbolos terminais por novos símbolos não-terminais e incorporação de novas produções da forma (2), uma para cada símbolo terminal de G_1;

iv. Substituição das produções $\alpha \rightarrow \beta$, em que $|\alpha| \leq |\beta|$, por um novo conjunto de produções das formas (3), (4), (5), (6) e (7);

v. Substituição das produções $\alpha \rightarrow \beta$, em que $|\alpha| > |\beta|$, por um novo conjunto de produções das formas (3), (4), (5), (6) e (7).

Algoritmo 7.4 (Forma normal para gramáticas irrestritas) *Obtenção de uma gramática irrestrita na forma normal.*

- Entrada: uma gramática irrestrita $G_1 = (V_1, \Sigma, P_1, S)$;
- Saída: uma gramática irrestrita $G_2 = (V_2, \Sigma, P_2, S)$, na forma normal, e tal que $L(G_2) = L(G_1)$;
- Método:

1. *Início*:
   Sendo $N_1 = V_1 - \Sigma$ e $N_2 = V_2 - \Sigma$, faz-se:
   - $N_2 \leftarrow N_1$
   - $P_2 \leftarrow \varnothing$
2. *Etapa (i)*:
   Eliminam-se as produções da forma $A \rightarrow \epsilon$ contidas em P_1:
   - $P_2 \leftarrow P_2 \cup \{XA \rightarrow X, AX \rightarrow X, \sigma A \rightarrow \sigma, A\sigma \rightarrow \sigma \mid A \rightarrow \epsilon \in P_1, X \in N_1, \sigma \in \Sigma\}$
3. *Etapa (ii)*:
   Incorporação da cadeia vazia, se for o caso:
   - $P_2 \leftarrow P_2 \cup \{S \rightarrow \epsilon$, se $\epsilon \in L(G_1)\}$
4. *Etapa (iii)*:
   Eliminação dos terminais originais e sua substituição por não-terminais correspondentes:
   - $N_2 \leftarrow N_2 \cup \{X_\sigma, \sigma \in \Sigma\}$
   - $P_2 \leftarrow P_2 \cup \{X_\sigma \rightarrow \sigma, \alpha_1 X_\sigma \alpha_2 \rightarrow \beta \mid \alpha_1 \sigma \alpha_2 \rightarrow \beta \in P_1, \sigma \in \Sigma, \alpha_1, \alpha_2 \in V_1^*\}$
   - $P_2 \leftarrow P_2 \cup \{X_\sigma \rightarrow \sigma, \alpha \rightarrow \beta_1 \sigma \beta_2 \mid \alpha \rightarrow \beta_1 \sigma \beta_2 \in P_1, \sigma \in \Sigma, \beta_1, \beta_2 \in V_1^*\}$
5. *Etapa (iv)*:
   Eliminação das produções $\alpha \rightarrow \beta, |\alpha| \leq |\beta|$:

– Para cada produção $\alpha \to \beta$ em P_1, com $|\alpha| \leq |\beta|$, tem-se:
$\alpha = A_1 A_2 ... A_m$
$\beta = B_1 B_2 ... B_n = B_1 B_2 ... B_m B_{m+1} ... B_n$, com $A_i, B_j \in N_1, n \geq m \geq 1$.

– Fazer:
$N_2 \leftarrow N_2 \cup \{X_i, 1 \leq i \leq n-1\}$
$P_2 \leftarrow P_2 \cup$

$$\begin{aligned}
\{A_1 &\to B_1 X_1 \\
X_1 A_2 &\to B_2 X_2 \\
X_2 A_3 &\to B_3 X_3 \\
&... \\
X_{m-2} A_{m-1} &\to B_{m-1} X_{m-1} \\
X_{m-1} A_m &\to B_m X_m \\
X_m &\to B_{m+1} X_{m+1} \\
X_{m+1} &\to B_{m+2} X_{m+2} \\
&... \\
X_{n-2} &\to B_{n-1} X_{n-1} \\
X_{n-1} &\to B_n\}
\end{aligned}$$

6. *Etapa (v)*: Eliminação das produções $\alpha \to \beta, |\alpha| > |\beta|$:

– Para cada produção $\alpha \to \beta$, com $|\alpha| > |\beta|$, tem-se:
$\alpha = A_1 A_2 ... A_n = A_1 A_2 ... A_m A_{m+1} ... A_n$
$\beta = B_1 B_2 ... B_m$, com $A_i, B_j \in N, n > m \geq 1$.

– Fazer:
$N_2 \leftarrow N_2 \cup \{X_i, 1 \leq i \leq n-m-1\} \cup \{Y_j, 0 \leq j \leq n-2\}$
$P_2 \leftarrow P_2 \cup$
$\{A_{n-1} A_n \to X_0 Y_0, X_0 Y_0 \to Y_0,$
$A_{n-2} Y_0 \to X_1 Y_1, X_1 Y_1 \to Y_1,$
$A_{n-3} Y_1 \to X_2 Y_2, X_2 Y_2 \to Y_2,$
...
$A_m Y_{n-m-2} \to X_{n-m-1} Y_{n-m-1}, X_{n-m-1} Y_{n-m-1} \to Y_{n-m-1} B_m,$
$A_{m-1} Y_{n-m-1} \to Y_{n-m} B_{m-1},$
$A_{m-2} Y_{n-m} \to Y_{n-m+1} B_{m-2},$
...
$A_1 Y_{n-2} \to Y_{n-2} B_1, Y_{n-2} B_1 \to B_1\}$

7. *Término*:
As produções do tipo $AB \to CD$ geradas nas etapas (iv) e (v) devem ainda ser substituídas por um conjunto de produções equivalente, porém em conformidade com as formas (3), (4), (5), (6) e (7):

$$N_2 \leftarrow N_2 \cup \{X, Y\}$$
$$P_2 \leftarrow P_2 \cup \{AB \rightarrow AY, AY \rightarrow XY, XY \rightarrow XD, XD \rightarrow CD\}$$

Todas as produções de G_2 estão em algum dos formatos permitidos na forma normal desejada, completando a prova. ■

Exemplo 7.5 De acordo com a etapa (iv), a produção $A_1A_2A_3 \rightarrow B_1B_2B_3B_4B_5B_6B_7$ deve ser substituída pelo conjunto de regras:

$$\begin{aligned}
\{A_1 &\rightarrow B_1X_1, && (7.1)\\
X_1A_2 &\rightarrow B_2X_2, && (7.2)\\
X_2A_3 &\rightarrow B_3X_3, && (7.3)\\
X_3 &\rightarrow B_4X_4, && (7.4)\\
X_4 &\rightarrow B_5X_5, && (7.5)\\
X_5 &\rightarrow B_6X_6, && (7.6)\\
X_6 &\rightarrow B_7\} && (7.7)
\end{aligned}$$

De fato, $A_1A_2A_3 \Rightarrow^+ B_1B_2B_3B_4B_5B_6B_7$, como se constata pela seguinte derivação:
$A_1A_2A_3 \Rightarrow B_1X_1A_2A_3 \Rightarrow B_1B_2X_2A_3 \Rightarrow B_1B_2B_3X_3 \Rightarrow$
$B_1B_2B_3B_4X_4 \Rightarrow B_1B_2B_3B_4B_5X_5 \Rightarrow B_1B_2B_3B_4B_5B_6X_6 \Rightarrow B_1B_2B_3B_4B_5B_6B_7$

Para completar, a produção (7.2) deve ainda ser substituída pelo conjunto:

$$\begin{aligned}
\{X_1A_2 &\rightarrow X_1N,\\
X_1N &\rightarrow MN,\\
MN &\rightarrow MX_2,\\
MX_2 &\rightarrow B_2X_2\}
\end{aligned}$$

Transformação semelhante deve ser aplicada na produção (7.3), pois as demais já se encontram enquadradas nos formatos originalmente propostos. □

Exemplo 7.6 De acordo com a etapa (v), a produção $A_1A_2A_3A_4A_5A_6A_7 \rightarrow B_1B_2B_3$ deve ser substituída pelo conjunto de produções:

$$\begin{aligned}
\{A_6A_7 &\rightarrow X_0Y_0, && (7.8)\\
X_0Y_0 &\rightarrow Y_0, && (7.9)\\
A_5Y_0 &\rightarrow X_1Y_1, && (7.10)\\
X_1Y_1 &\rightarrow Y_1, && (7.11)\\
A_4Y_1 &\rightarrow X_2Y_2, && (7.12)\\
X_2Y_2 &\rightarrow Y_2, && (7.13)\\
A_3Y_2 &\rightarrow X_3Y_3, && (7.14)\\
X_3Y_3 &\rightarrow Y_3B_3, && (7.15)\\
A_2Y_3 &\rightarrow X_4Y_4, && (7.16)\\
X_4Y_4 &\rightarrow Y_4B_2, && (7.17)\\
A_1Y_4 &\rightarrow Y_5B_1, && (7.18)\\
Y_5B_1 &\rightarrow B_1\} && (7.19)
\end{aligned}$$

De fato, $A_1A_2A_3A_4A_5A_6A_7 \Rightarrow^+ B_1B_2B_3$, como se constata pela seguinte derivação:
$A_1A_2A_3A_4A_5A_6A_7 \Rightarrow A_1A_2A_3A_4A_5X_0Y_0 \Rightarrow A_1A_2A_3A_4A_5Y_0 \Rightarrow A_1A_2A_3A_4X_1Y_1 \Rightarrow$
$A_1A_2A_3A_4Y_1 \Rightarrow A_1A_2A_3X_2Y_2 \Rightarrow A_1A_2A_3Y_2 \Rightarrow A_1A_2X_3Y_3 \Rightarrow A_1A_2Y_3B_3 \Rightarrow$

$A_1 X_4 Y_4 B_3 \Rightarrow A_1 Y_4 B_2 B_3 \Rightarrow Y_5 B_1 B_2 B_3 \Rightarrow B_1 B_2 B_3$

Para completar, a produção (7.8) deve ainda ser substituída pelo conjunto:

$$\begin{array}{rcl} \{A_6 A_7 & \rightarrow & A_6 N, \\ A_6 N & \rightarrow & MN, \\ MN & \rightarrow & MY_0, \\ MY_0 & \rightarrow & X_0 Y_0\} \end{array}$$

Transformações semelhantes devem ser aplicadas às produções (7.10), (7.12), (7.14), (7.15), (7.16), (7.17) e (7.18). Assim, todas as produções resultantes se enquadram nas formas originalmente propostas. □

7.5 Equivalência entre Gramáticas Irrestritas e Linguagens Recursivamente Enumeráveis

As Máquinas de Turing com fita infinita definem a classe das linguagens recursivamente enumeráveis, que também podem ser geradas pelas gramáticas irrestritas. Tal resultado pode ser comprovado através dos dois teoremas seguintes, que demonstram a equivalência dos dois tipos de dispositivos, no que se refere à classe de linguagens que eles são capazes de representar. A demonstração original é devida a Chomsky em [62].

Teorema 7.4 (Irrestritas ⇒ recursivamente enumeráveis) *Seja $L = L(G)$, sendo G uma gramática irrestrita. Então L é uma linguagem recursivamente enumerável.*

Justificativa A demonstração é feita a partir da construção de uma Máquina de Turing M não-determinística que aceita L. M é construída de tal modo que seus movimentos simulam as derivações das sentenças geradas por G. A máquina M que reconhece $L(G)$ apresenta três fitas de trabalho:

- Na primeira fita está inicialmente gravada a cadeia w que se deseja examinar para determinar se ela pertence ou não a L;
- Na segunda fita fica gravada uma codificação $C(G)$ da gramática irrestrita G;
- Na terceira fita está inicialmente gravado, na primeira posição, o símbolo não-terminal S, a raiz da gramática G. A cadeia gravada na terceira fita será denotada por z e, portanto, inicialmente $|z| = |S| = 1$.

A operação de M é não-determinística e acontece da seguinte forma:

1. Determinar todos os pares (i, α) tais que, a partir da posição i da cadeia z, seja possível identificar uma subcadeia α, tal que $\alpha \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n \in P$, sendo P o conjunto das regras de G gravado na segunda fita de trabalho;
2. Para cada par (i, α) obtido no passo (1), criar um novo "thread" para M, e deslocar o respectivo cursor de acesso da fita número 3 até a posição i; encerrar o "thread" original na configuração de rejeição;
3. Para cada "thread" criado no passo (2):

a) Criar n novos "threads" para M, sendo que cada "thread" j, $1 \leq j \leq n$, deve substituir, na cadeia z, a partir da posição i, a subcadeia α pela subcadeia β_j; encerrar o "thread" original na configuração de rejeição;

b) Para cada "thread" criado no passo (3.a):

   i. Se $z \neq w$, desviar para o passo (1);

   ii. Se $z = w$, encerrar o "thread" na configuração de aceitação.

■

Note-se que a geração de uma nova forma sentencial na fita número 3 pode provocar a manutenção, a redução ou o aumento de seu comprimento, em relação à forma sentencial imediatamente anterior. Dessa forma, M deve ser capaz de efetuar a contração ou a expansão da cadeia que representa cada forma sentencial, evitando sua fragmentação ou perda de informação.

A máquina M assim construída simula, na fita número 3, as derivações de G. A escolha da produção a ser aplicada em cada momento, no entanto, é não-determinística. Se $w \in L$, é certo que existe pelo menos uma derivação que a gera. Logo, é fato que alguma instância de M inevitavelmente reproduzirá essa derivação, o que resultará na aceitação de w. Se, por outro lado, $w \notin L$, isso significa que não existe derivação possível de w em G. Logo, nenhuma das instâncias de M será capaz de atingir uma configuração final. A cadeia de entrada será rejeitada, ou então M iniciará uma seqüência interminável de movimentações. Portanto, $L(M) = L(G)$, e L é uma linguagem recursivamente enumerável.

Cumpre notar a semelhança entre a técnica descrita no Teorema 7.4 e o Algoritmo 5.2, que mostra como obter uma Máquina de Turing com fita limitada a partir de uma gramática sensível ao contexto. A diferença é que, naquele caso, as formas sentenciais geradas possuem comprimento não-decrescente, e isso facilita a sua comparação sistemática com a cadeia de entrada. Além disso, não existem garantias, como no caso do Algoritmo 5.2, de que todos os "threads" se encerram após um tempo finito de execução — exceto, naturalmente, para o "thread" que gera a seqüência de derivações que resulta na cadeia de entrada, caso essa seja uma sentença da linguagem gerada por G. Ao contrário, é possível que, para cadeias não pertencentes à linguagem, existam "threads" de M executando seqüências intermináveis de movimentações, sem que outros "threads" atinjam uma configuração de aceitação. Por esse motivo, a técnica descrita no Teorema 7.4 não recebe a designação de "algoritmo".

Teorema 7.5 (Irrestritas $\Leftarrow$ recursivamente enumeráveis) *Seja $L = L(M)$, sendo M uma Máquina de Turing com fita infinita. Então $L = L(G)$, sendo G uma gramática irrestrita.*

Justificativa Segue a mesma linha adotada na demonstração do Teorema 5.3 e consiste na obtenção de uma gramática irrestrita que reproduz, na derivação de suas sentenças, os movimentos da Máquina de Turing M que aceita a mesma linguagem. Se a cadeia de entrada é aceita por M, ela também é gerada pela gramática. Cadeias não aceitas por M não são geradas pela gramática.

O Algoritmo 7.5 descrito a seguir especifica os passos que devem ser executados na transformação sistemática de uma Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, <, B, F)$ em uma gramática irrestrita $G = (V, \Sigma, P, S)$.

Algoritmo 7.5 (Irrestritas $\Leftarrow$ recursivamente enumeráveis) *Obtenção de uma gramática irrestrita a partir de uma Máquina de Turing.*

- Entrada: uma Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, <, B, F)$ com fita infinita;
- Saída: uma gramática irrestrita $G = (V, \Sigma, P, S)$ tal que $L(G) = L(M)$;
- Método:

  Inicialmente são gerados os símbolos não-terminais de G. Esses símbolos possuem o formato geral $[\sigma, \alpha]$, onde $\sigma \in \Sigma, \alpha$ é uma cadeia sobre $\{\sigma, q, <, \#\}$ e $q \in Q$. A primeira destas duas componentes do símbolo — $[\sigma, ...]$ — representa um certo terminal da cadeia de entrada. Sua segunda componente — $[..., \alpha]$ — representa parte da configuração assumida pela Máquina de Turing no reconhecimento da referida cadeia de entrada.

  Os conjuntos P e S de G são inicialmente definidos como:

  I. $P \leftarrow \{S \rightarrow [\sigma_i, < q_0\sigma_i]A, \forall\, \sigma_i \in \Sigma\}$
  $N \leftarrow \{S, A, [\sigma_i, < q_0\sigma_i], \forall\, \sigma_i \in \Sigma\}$

  II. $P \leftarrow P \cup \{S \rightarrow [\sigma_i, < q_0\sigma_i\#], \forall\, \sigma_i \in (\Sigma \cup \{B\})\}$
  $N \leftarrow N \cup \{[\sigma_i, < q_0\sigma_i\#], \forall\, \sigma_i \in (\Sigma \cup \{B\})\}$

  III. $P \leftarrow P \cup \{A \rightarrow [\sigma_i, \sigma_i]A, A \rightarrow [\sigma_i, \sigma_i\#], \forall\, \sigma_i \in \Sigma\}$
  $N \leftarrow N \cup \{[\sigma_i, \sigma_i], [\sigma_i, \sigma_i\#], \forall\, \sigma_i \in \Sigma\}$

  Eles permitem gerar formas sentenciais dos formatos seguintes:

  (a) $[B, < q_0B\#]$, ou

  (b) $[\sigma_i, < q_0\sigma_i\#]$, ou

  (c) $[\sigma_i, < q_0\sigma_i][\sigma_j, \sigma_j][\sigma_k, \sigma_k]...[\sigma_u, \sigma_u\#]$

  Em todos os casos, o símbolo "#" é usado para delimitar, nas formas sentenciais geradas pela gramática, o término da cadeia gravada sobre a fita de trabalho de M. O símbolo "#" pode ser deslocado à direita, e neste caso símbolos "B" são adicionados ao final da cadeia de entrada. A cadeia de entrada vazia é representada pela forma sentencial do tipo (a). As cadeias de entrada não-vazias são representadas pelas formas dos tipos (b) (cadeia unitária) e (c) (cadeia de comprimento maior ou igual a dois).

  Assim como na demonstração do Teorema 5.3, essas formas sentenciais devem ser interpretadas como a representação de duas componentes:

  – $[\sigma_i, ...][\sigma_j, ...][\sigma_k, ...]...[\sigma_u, ...] = \sigma_i\sigma_j\sigma_k...\sigma_u$, correspondente à cadeia de entrada que se pretende verificar em M;
  e

– $[..., < q_0\sigma_i][..., \sigma_j][..., \sigma_k]...[..., \sigma_u\#] =< q_0\sigma_i\sigma_j\sigma_k...\sigma_u\#$, correspondente à configuração inicial de M com a cadeia de entrada acima.

Um conjunto adicional de regras de P é especificado, com a finalidade de garantir que, qualquer que seja a movimentação efetuada por M, a partir de uma configuração qualquer, haja sempre uma possibilidade correspondente de derivação em G.

As transições de M enquadram-se em três casos:

– *Caso (i)*: $\delta(q_i, <) = (q_j, <, D)$

– *Caso (ii)*: $\delta(q_i, \sigma_m) = (q_j, \sigma_n, D)$

– *Caso (iii)*: $\delta(q_i, \sigma_m) = (q_j, \sigma_n, E)$

O conjunto das produções de G que são capazes de promover tais derivações é, naturalmente, obtido a partir da especificação da função δ. Deve-se, portanto, considerar cada um dos três casos indicados. A partir de cada um deles, novas produções são formuladas e, a partir destas, novos símbolos não-teminais são acrescentados à gramática. Observe-se que nos casos (*ii*) e (*iii*) as transições são feitas com símbolos do alfabeto Γ, ou seja, $\sigma_m, \sigma_n \in \Gamma$, o que inclui não apenas os símbolos originalmente empregados na composição da cadeia de entrada, mas também símbolos gravados por M na fita de trabalho durante o seu próprio processamento.

– *Caso (i)*: $\delta(q_i, <) = (q_j, <, D)$

IV. $P \leftarrow P \cup \{[\sigma_m, q_i < \sigma_m] \rightarrow [\sigma_m, < q_j\sigma_m], \forall\, \sigma_m \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_m, q_i < \sigma_m], \forall\, \sigma_m \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_m, < q_j\sigma_m], \forall\, \sigma_m \in \Gamma\}$

– *Caso (ii)*: $\delta(q_i, \sigma_m) = (q_j, \sigma_n, D)$

O símbolo corrente (σ_m) pode ser:

* o primeiro da cadeia de entrada:
  - cadeia unitária: VIII;
  - cadeia de comprimento dois: IX;
  - cadeia de comprimento maior que dois: X;
* o último da cadeia de entrada:
  - cadeia unitária: VIII;
  - cadeia não-unitária: XI;

- o penúltimo da cadeia de entrada:
  - cadeia de comprimento dois: IX;
  - cadeia de comprimento maior que dois: XII;
- ou ainda estar em qualquer posição diferente destas:
  - XIII;

VIII. $P \leftarrow P \cup \{[\sigma_p, < q_i\sigma_m\#] \rightarrow [\sigma_p, < \sigma_n][B, q_jB\#], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < q_i\sigma_m\#], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[B, q_jB\#]\}$

IX. $P \leftarrow P \cup \{[\sigma_p, < q_i\sigma_m][\sigma_q, \sigma_r\#] \rightarrow [\sigma_p, < \sigma_n][\sigma_q, q_j\sigma_r\#],$
$\forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_q, \sigma_r\#], \forall\, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_q, q_j\sigma_r\#], \forall\, \sigma_q, \sigma_r \in \Gamma\}$

X. $P \leftarrow P \cup \{[\sigma_p, < q_i\sigma_m][\sigma_q, \sigma_r] \rightarrow [\sigma_p, < \sigma_n][\sigma_q, q_j\sigma_r],$
$\forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_q, \sigma_r], \forall\, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_q, q_j\sigma_r], \forall\, \sigma_q, \sigma_r \in \Gamma\}$

XI. $P \leftarrow P \cup \{[\sigma_p, q_i\sigma_m\#] \rightarrow [\sigma_p, \sigma_n][B, q_jB\#], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_i\sigma_m\#], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[B, q_jB\#]\}$

XII. $P \leftarrow P \cup \{[\sigma_p, q_i\sigma_m][\sigma_q, \sigma_r\#] \rightarrow [\sigma_p, \sigma_n][\sigma_q, q_j\sigma_r\#],$
$\forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_q, \sigma_r\#], \forall\, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_q, q_j\sigma_r\#], \forall\, \sigma_q, \sigma_r \in \Gamma\}$

XIII. $P \leftarrow P \cup \{[\sigma_p, q_i\sigma_m][\sigma_q, \sigma_r] \rightarrow [\sigma_p, \sigma_n][\sigma_q, q_j\sigma_r],$
$\forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_q, \sigma_r], \forall\, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_q, q_j\sigma_r], \forall\, \sigma_q, \sigma_r \in \Gamma\}$

– *Caso (iii)*: $\delta(q_i, \sigma_m) = (q_j, \sigma_n, E)$

O símbolo corrente (σ_m) pode:

- ser o primeiro da cadeia de entrada:
  - cadeia unitária: XIV;
  - cadeia não-unitária: XV;
- ser o último da cadeia de entrada:
  - cadeia unitária: XIV;
  - cadeia de comprimento dois: XVI;
  - cadeia de comprimento maior que dois: XVII;
- ser o segundo da cadeia de entrada:
  - cadeia de comprimento dois: XVI;
  - cadeia de comprimento maior que dois: XVIII;
- estar em qualquer posição diferente destas:
  - XIX;

XIV. $P \leftarrow P \cup \{[\sigma_p, < q_i\sigma_m\#] \rightarrow [\sigma_p, q_j < \sigma_n\#], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < q_i\sigma_m\#], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_j < \sigma_n\#], \forall\, \sigma_p \in \Gamma\}$

XV. $P \leftarrow P \cup \{[\sigma_p, < q_i\sigma_m] \rightarrow [\sigma_p, q_j < \sigma_n], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_j < \sigma_n], \forall\, \sigma_p \in \Gamma\}$

XVI. $P \leftarrow P \cup \{[\sigma_p, < \sigma_q][\sigma_r, q_i\sigma_m\#] \rightarrow [\sigma_p, < q_j\sigma_q][\sigma_r, \sigma_n\#], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < \sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_r, q_i\sigma_m\#], \forall\, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < q_j\sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_r, \sigma_n\#], \forall\, \sigma_r \in \Gamma\}$

XVII. $P \leftarrow P \cup \{[\sigma_p, \sigma_q][\sigma_r, q_i\sigma_m\#] \rightarrow [\sigma_p, q_j\sigma_q][\sigma_r, \sigma_n\#], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, \sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_r, q_i\sigma_m\#], \forall\, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_j\sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_r, \sigma_n\#], \forall\, \sigma_r \in \Gamma\}$

XVIII. $P \leftarrow P \cup \{[\sigma_p, < \sigma_q][\sigma_r, q_i\sigma_m] \rightarrow [\sigma_p, < q_j\sigma_q][\sigma_r, \sigma_n], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < \sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$

$N \leftarrow N \cup \{[\sigma_r, q_i\sigma_m], \forall\, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, < q_j\sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_r, \sigma_n], \forall\, \sigma_r \in \Gamma\}$

XIX. $P \leftarrow P \cup \{[\sigma_p, \sigma_q][\sigma_r, q_i\sigma_m] \rightarrow [\sigma_p, q_j\sigma_q][\sigma_r, \sigma_n], \forall\, \sigma_p, \sigma_q, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, \sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_r, q_i\sigma_m], \forall\, \sigma_r \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_j\sigma_q], \forall\, \sigma_p, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_r, \sigma_n], \forall\, \sigma_r \in \Gamma\}$

A operação de M cessa ao ser atingida uma configuração para a qual não exista movimentação possível, ou seja, quando $\delta(q_i, \sigma_m) = \varnothing$. Nesta situação, se $q_i \in F$, diz-se que M aceita a cadeia de entrada.

Condição semelhante deve ser alcançada através de G. Suponha-se que, quando o autômato pára de se movimentar, a configuração seja $(\alpha, q_i\sigma_m\beta)$. A forma sentencial equivalente em G será [...][...]...[σ_n, $q_i\sigma_m$][...].

Se, nesta situação, $q_i \in F$ e $\delta(q_i, \sigma_m) = \varnothing$, então as seguintes regras devem ser adicionadas à gramática G:

- *Caso (iv)*: $q_i \in F$ e $\delta(q_i, <) = \varnothing$

XX. $P \leftarrow P \cup \{[\sigma_p, q_i < \sigma_q] \rightarrow \sigma_p, \forall\, \sigma_p \in (\Sigma \cup \{B\}), \forall\, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_i < \sigma_q], \forall\, \sigma_p \in (\Sigma \cup \{B\}), \forall\, \sigma_q \in \Gamma\}$

XXI. $P \leftarrow P \cup \{[\sigma_p, q_i < \sigma_q\#] \rightarrow \sigma_p\#, \forall\, \sigma_p \in (\Sigma \cup \{B\}), \forall\, \sigma_q \in \Gamma\}$
$N \leftarrow N \cup \{[\sigma_p, q_i < \sigma_q\#], \forall\, \sigma_p \in (\Sigma \cup \{B\}), \forall\, \sigma_q \in \Gamma\}$

- *Caso (v)*: $q_i \in F$ e $\delta(q_i, \sigma_m) = \varnothing$

XXII. $P \leftarrow P \cup \{[\sigma_p, < q_i\sigma_m] \rightarrow \sigma_p, \forall\, \sigma_p \in (\Sigma \cup \{B\})\}$
$N \leftarrow N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in (\Sigma \cup \{B\})\}$

XXIII. $P \leftarrow P \cup \{[\sigma_p, < q_i\sigma_m\#] \rightarrow \sigma_p\#, \forall\, \sigma_p \in (\Sigma \cup \{B\})\}$
$N \leftarrow N \cup \{[\sigma_p, < q_i\sigma_m], \forall\, \sigma_p \in (\Sigma \cup \{B\})\}$

XXIV. $P \leftarrow P \cup \{[\sigma_p, q_i\sigma_m\#] \rightarrow \sigma_p\#, \forall\, \sigma_p \in (\Sigma \cup \{B\})\}$
$N \leftarrow N \cup \{[\sigma_p, q_i\sigma_m], \forall\, \sigma_p \in (\Sigma \cup \{B\})\}$

XXV. $P \leftarrow P \cup \{[\sigma_p, q_i\sigma_m] \rightarrow \sigma_p, \forall\, \sigma_p \in (\Sigma \cup \{B\})\}$
$N \leftarrow N \cup \{[\sigma_p, q_i\sigma_m], \forall\, \sigma_p \in (\Sigma \cup \{B\})\}$

Essas produções dão início ao processo de conversão da cadeia armazenada no lado esquerdo dos não-terminais — e até então intocada — em uma sentença a ser efetivamente gerada por G. A forma sentencial inicial é obtida através da derivação [...] [...] ... [σ_n, $q_i\sigma_m$] [...] $\Rightarrow$ [...] [...] ... σ_n [...], se $q_i \in F$ e $\delta(q_i, \sigma_m) = \varnothing$.

Cumpre notar, caso a configuração final de M seja atingida em uma situação que corresponda, em G, à aplicação de alguma das produções geradas em XXI, XXIII e XXIV, que o símbolo "#", originalmente gravado "dentro" de algum não-terminal

de gramática (isto é, delimitado pelos símbolos "[" e "]"), é transposto para "fora" do mesmo (sendo que, neste caso, os delimitadores "[" e "]", e conseqüentemente o próprio não-terminal, deixam de existir), passando a indicar o término da cadeia de entrada.

A partir deste ponto, devem ser acrescentadas produções em P de tal modo que os não-terminais remanescentes na forma sentencial gerada por G sejam todos substituídos por símbolos terminais, conforme o valor armazenado no lado esquerdo, no interior do mesmo ($[\sigma, ...]$):

XXVI. $P \leftarrow P \cup \{[\sigma_m, \alpha]\sigma_n \rightarrow \sigma_m\sigma_n, \forall\, \sigma_n \in (\Sigma \cup \{B\}), \forall [\sigma_m, \alpha] \in V\}$

XXVII. $P \leftarrow P \cup \{\sigma_n[\sigma_m, w] \rightarrow \sigma_n\sigma_m, \forall\, \sigma_n \in (\Sigma \cup \{B\}), \forall [\sigma_m, w] \in V$, desde que w não contenha o símbolo "#"$\}$

XXVIII. $P \leftarrow P \cup \{\sigma_n[\sigma_m, w] \rightarrow \sigma_n\sigma_m\#, \forall\, \sigma_n \in (\Sigma \cup \{B\}), \forall [\sigma_m, w] \in V$, desde que w contenha o símbolo "#"$\}$

Nesta situação, a forma sentencial produzida por G é da forma $\alpha\beta\#$, onde $\alpha \in \Sigma^*$ é cadeia de entrada originalmente submetida à Máquina de Turing, e $\beta \in \{B\}^*$ representa símbolos B que foram adicionados ao final da cadeia de entrada pelo deslocamento para a direita do cursor de acesso à fita de trabalho. A eliminação da subcadeia $\beta\#$ é feita por intermédio das produções:

XXIX. $P \leftarrow P \cup \{B\# \rightarrow \#\}$

XXX. $P \leftarrow P \cup \{\sigma_i\# \rightarrow \sigma_i, \forall\, \sigma_i \in \Sigma\}$

A inspeção das produções geradas nos itens I a XXX (em particular nos itens XXIX e XXX) permite concluir que toda e qualquer gramática gerada por esse método é uma gramática irrestrita. Será omitida a demonstração formal da equivalência das Máquinas de Turing com as gramáticas irrestritas assim construídas. Tal demonstração pode, no entanto, ser encontrada em [52]. ■

Exemplo 7.7 Seja M tal que:

$$\begin{aligned}
\Sigma &= \{x, y, z\} \\
\Gamma &= \{x, y, z, X, Y, Z, B\} \\
\delta &\supset \{(q_0, x) \rightarrow (q_1, X, D), (q_1, y) \rightarrow (q_2, Y, D), (q_2, z) \rightarrow (q_3, Z, D), \\
&\quad (q_3, B) \rightarrow (q_4, x, D), (q_4, B) \rightarrow (q_5, B, D), (q_5, B) \rightarrow (q_6, y, D), \\
&\quad (q_6, B) \rightarrow (q_7, B, D), (q_7, B) \rightarrow (q_8, z, D), (q_8, B) \rightarrow (q_9, B, E), \\
&\quad (q_9, z) \rightarrow (q_{10}, z, E)\} \\
\delta(q_{10}, B) &= \varnothing \\
\{q_{10}\} &\subset F
\end{aligned}$$

A seqüência de movimentos abaixo ilustra a aceitação da cadeia xyz por M:

$(<, q_0xyz) \vdash (< X, q_1yz) \vdash (< XY, q_2z) \vdash (< XYZ, q_3B) \vdash$
$(< XYZx, q_4B) \vdash (< XYZxB, q_5B) \vdash (< XYZxBy, q_6B) \vdash (< XYZxByB, q_7B) \vdash$
$(< XYZxByBz, q_8B) \vdash (< XYZxByB, q_9zB) \vdash (< XYZxBy, q_{10}BzB)$

A seqüência de derivações abaixo mostra as formas sentenciais geradas por uma gramática irrestrita G, correspondentes aos movimentos de M acima. As produções e os símbolos não-terminais utilizados foram derivados diretamente da especificação de M, conforme o Algoritmo 7.5. Note-se que $S \Rightarrow^* xyz$, e portanto $xyz \in L(G)$.

$$
\begin{aligned}
&S \Rightarrow \\
&[x, < q_0 x]A \Rightarrow \\
&[x, < q_0 x][y, y]A \Rightarrow \\
&[x, < q_0 x][y, y][z, z\#] \Rightarrow \\
&[x, < X][y, q1y][z, z\#] \Rightarrow \\
&[x, < X][y, Y][z, q_2 z\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, q_3 B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, q_4 B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B][B, q_5 B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B][B, y][B, q_6 B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B][B, y][B, B][B, q_7 B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B][B, y][B, B][B, z][B, q_8 B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B][B, y][B, B][B, q_9 z][B, B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B][B, y][B, q_{10} B][B, z][B, B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B][B, y]B[B, z][B, B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B]BB[B, z][B, B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x][B, B]BBB[B, B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x]BBBB[B, B\#] \Rightarrow \\
&[x, < X][y, Y][z, Z][B, x]BBBBB\# \Rightarrow \\
&[x, < X][y, Y][z, Z]BBBBBB\# \Rightarrow \\
&[x, < X][y, Y]zBBBBBB\# \Rightarrow \\
&[x, < X]yzBBBBBB\# \Rightarrow \\
&xyzBBBBBB\# \Rightarrow \\
&xyzBBBBB\# \Rightarrow \\
&xyzBBBB\# \Rightarrow \\
&xyzBBB\# \Rightarrow \\
&xyzBB\# \Rightarrow \\
&xyzB\# \Rightarrow \\
&xyz\# \Rightarrow \\
&xyz
\end{aligned}
$$

□

7.6 Relação entre Linguagens Recursivamente Enumeráveis e Linguagens Recursivas

Os dois teoremas seguintes estabelecem a relação de inclusão própria que existe entre as linguagens recursivas e as recursivamente enumeráveis.

Teorema 7.6 (Recursivas $\subseteq$ recursivamente enumeráveis) *Toda linguagem recursiva é também uma linguagem recursivamente enumerável.*

Justificativa As classes de linguagens recursivas e recursivamente enumeráveis foram definidas como sendo aquelas cujas linguagens são aceitas por pelo menos uma Máquina

de Turing. As linguagens recursivas, em particular, além de serem aceitas, são também decididas por pelo menos uma Máquina de Turing, o que não é sempre verdade para as linguagens estritamente recursivamente enumeráveis.

Portanto, toda Máquina de Turing que decide uma linguagem também aceita a mesma linguagem. No entanto, linguagens aceitas por uma Máquina de Turing podem não ser decidíveis por nenhuma Máquina de Turing. O critério de decisão é mais forte do que o de aceitação. Logo, toda linguagem recursiva é também recursivamente enumerável. ■

Teorema 7.7 (Recursivas $\neq$ recursivamente enumeráveis) *A classe das linguagens recursivas constitui subconjunto próprio da classe das linguagens recursivamente enumeráveis.*

Justificativa A linguagem L_U, conforme demonstrado anteriormente no Teorema 6.4, é uma linguagem não-recursiva. A demonstração de que L_U é uma linguagem recursivamente enumerável, feita a seguir, basta para provar que a classe das linguagens recursivas constitui subconjunto próprio da classe das linguagens recursivamente enumeráveis.

$$L_U = \{C(M)w \in \Sigma^* \mid w \in L(M)\}$$

onde

- $C(M)$ é a codificação de uma Máquina de Turing com fita limitada, cujo alfabeto de entrada é Σ, como cadeia sobre o próprio alfabeto Σ;
- $w \in \Sigma^*$ é uma cadeia de entrada qualquer de M;

Para provar que L_U é recursivamente enumerável, basta mostrar que L_U é aceita por uma Máquina de Turing N, a qual pode ser obtida conforme o Algoritmo 7.6.

Algoritmo 7.6 (Máquina de Turing para L_U) *Obtenção de uma Máquina de Turing que aceita a linguagem não-recursiva L_U.*

- Entrada: a linguagem L_U;
- Saída: uma Máquina de Turing N que aceita L_U;
- Método:

    1. N simula a operação da Máquina M no processamento da cadeia de entrada w, onde $C(M)w$ é a cadeia de entrada de N. N possui duas fitas: na primeira é gravada a cadeia $\gamma = C(M)w$ que se pretende analisar; a segunda é usada como área de trabalho de N, e servirá para guardar, em cada instante, o estado corrente de M;
    2. Três possibilidades podem acontecer durante o processamento de N:
        - M aceita w. Neste caso, N deverá encerrar sua operação aceitando $C(M)w$;
        - M rejeita w. Neste caso, N deverá encerrar sua operação rejeitando $C(M)w$;

– M inicia uma seqüência infindável de movimentações com w. Neste caso, N inevitavelmente também iniciará, correspondentemente, com $C(M)w$, uma seqüência interminável de movimentações.

Assim, é fácil perceber que todas as cadeias que representam pares (M, w), em que $w \in L(M)$, serão também aceitas por N. Já as cadeias que representam pares (M, w), em que $w \notin L(M)$, farão com que N as rejeite ou inicie uma seqüência infinita de movimentações. Portanto, L_U é uma linguagem recursivamente enumerável.

Também a linguagem $L_K = \{w_i \mid w_i \in L(M_i)\}$, definida e estudada no Teorema 6.5, constitui exemplo de linguagem recursivamente enumerável, porém não-recursiva.

Esta linguagem é aceita pela Máquina de Turing M, construída de acordo com o Algoritmo 7.7.

Algoritmo 7.7 (Máquina de Turing para L_K) *Obtenção de uma Máquina de Turing que aceita a linguagem não-recursiva L_K.*

- Entrada: a linguagem L_K;
- Saída: uma Máquina de Turing N que aceita L_K;
- Método:

1. A fita de entrada de N contém, originalmente, a cadeia $w \in \Sigma^*$ a ser analisada;
2. $i \leftarrow 0$;
3. N gera a cadeia α_i;
4. Para cada cadeia α_i gerada, N compara α_i com w;
5. Se α_i for igual a w, então:
   a) N constrói a Máquina de Turing M_i correspondente à cadeia α_i e armazena $C(M_i)$ na fita de trabalho;
   b) N simula M_i com a entrada α_i;
   c) Se M_i aceita α_i, então N aceita w;
6. $i \leftarrow i + 1$;
7. Desviar para (3).

N gera, uma por uma, porém uma de cada vez, todas as cadeias $\alpha_i \in \Sigma^*$, ordenadas lexicograficamente. Uma vez identificado o valor de "i" para a cadeia de entrada fornecida (o que acontece após um tempo finito de processamento), N gera M_i e verifica se M_i aceita α_i. N aceita a cadeia de entrada w se e apenas se $\alpha_i(w)$ for aceita por M_i. Logo, L_K é recursivamente enumerável. ■

7.7 Linguagens que não são Recursivamente Enumeráveis

A classe das linguagens recursivamente enumeráveis não corresponde, como se poderia supor, ao conjunto mais abrangente de linguagens que pode ser construído sobre um alfabeto finito qualquer.

Conforme será demonstrado a seguir, existem inúmeras linguagens para as quais não é possível definir Máquinas de Turing que as aceitem ou gramáticas que as gerem. O conjunto das linguagens recursivamente enumeráveis é, na verdade, um subconjunto próprio e de cardinalidade inferior ao conjunto de todas as linguagens que podem ser definidas sobre um alfabeto finito, qualquer que seja ele. Pelos mesmos motivos, verifica-se que esta nova e mais abrangente classe de linguagens é constituída por linguagens que não possuem formalização gramatical ([56]).

O conjunto das linguagens formadas sobre alfabetos e que não possuem gramáticas (e conseqüentemente Máquinas Turing) correspondentes será denominado de classe das **linguagens não-gramaticais** ou **não-recursivamente enumeráveis**. A Figura 7.1 ilustra essas relações:

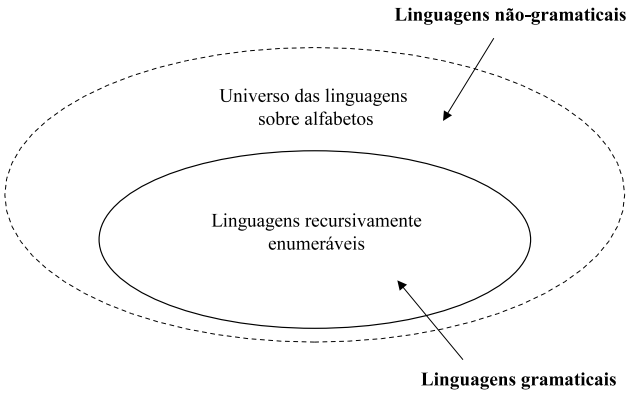

Figura 7.1: Linguagens gramaticais e não-gramaticais

Na seqüência é demonstrada a existência de linguagens não-gramaticais. Antes, porém, é necessário demonstrar os Teoremas auxiliares 7.8, 7.9 e 7.10.

Teorema 7.8 (Máquinas de Turing são enumeráveis) *O conjunto* $C_1 = \{M \mid M$ *é uma Máquina de Turing e* $L(M) \subseteq \Sigma^*\}$ *é enumerável.* C_1 *corresponde ao conjunto de todas as Máquinas de Turing definíveis sobre um certo alfabeto* Σ*.*

Justificativa Uma Máquina de Turing é representada através de uma cadeia de símbolos de comprimento finito. O alfabeto utilizado pode ser amplo, como, por exemplo:

$$\{a, b, c, ...z, 0, 1, 2...9, (,), \{, \}, \rightarrow, ...\}$$

utilizado na maioria das representações algébricas construídas até o momento, ou, ainda, um alfabeto restrito, como, por exemplo, o alfabeto:

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

empregado na demonstração do Teorema 6.4.

Qualquer que seja o alfabeto finito Σ utilizado, sabe-se que o conjunto Σ^* é enumerável. Uma maneira fácil de perceber isso é considerar a ordenação lexicográfica das cadeias geradas sobre Σ, conforme foi feito na demonstração do Teorema 7.2. Para gerar tal ordenação, é suficiente convencionar uma relação de ordem entre os elementos de Σ. O estabelecimento de uma bijeção entre o conjunto dos números naturais $\mathbb{N}$ e o conjunto Σ^* é imediato. Logo, Σ^* é enumerável.

Por outro lado, nem toda cadeia sobre Σ representa uma Máquina de Turing válida. O conjunto que interessa a este estudo é, portanto, um subconjunto de Σ^*. Conforme o Teorema 1.4, qualquer subconjunto de um conjunto enumerável é também enumerável. Logo, C_1 é um conjunto enumerável. ■

Teorema 7.9 (Gramáticas são enumeráveis) *O conjunto $C_2 = \{G \mid G$ é uma gramática e $L(G) \subseteq \Sigma^*\}$ é enumerável. C_2 corresponde ao conjunto de todas as gramáticas definíveis sobre um certo alfabeto Σ.*

Justificativa A demonstração é feita de maneira análoga à do Teorema 7.8 ■

Teorema 7.10 (O conjunto de todas as linguagens não é enumerável) *O conjunto $C_3 = 2^{\Sigma^*}$ não é enumerável. C_3 corresponde ao conjunto de todas as linguagens definíveis sobre um certo alfabeto Σ.*

Justificativa O Teorema 1.3 garante que, para qualquer conjunto A, finito ou infinito, a cardinalidade de A é sempre inferior à cardinalidade de 2^A. Logo, a cardinalidade de Σ^* é menor do que a cardinalidade de 2^{Σ^*}. ■

A demonstração da existência de linguagens não-gramaticais, e portanto da relação de inclusão própria da classe das linguagens recursivamente enumeráveis com a classe das linguagens formadas sobre alfabetos, é feita a seguir.

Teorema 7.11 (Linguagens não-recursivamente enumeráveis) *Existem linguagens que não são recursivamente enumeráveis.*

Justificativa Como a cardinalidade de C_3 é maior do que a cardinalidade de C_1 e de C_2, que por sua vez possuem a mesma cardinalidade, é fácil perceber que não existe relação um-para-um (função bijetora) entre os elementos desses conjuntos ($C_1 \leftrightarrow C_3$ ou $C_2 \leftrightarrow C_3$). Portanto, existem inúmeras linguagens para as quais não há Máquina de Turing nem gramática irrestrita correspondente, e portanto o conjunto das linguagens recursivamente enumeráveis constitui subconjunto próprio do conjunto de todas as linguagens que podem ser definidas sobre um dado alfabeto. A correspondência entre as gramáticas irrestritas e as Máquinas de Turing foi estabelecida nos Teoremas 7.4 e 7.5.

Assim, o conjunto das linguagens não-gramaticais possui cardinalidade maior do que a cardinalidade do conjunto das linguagens recursivamente enumeráveis. Apesar de ambos os conjuntos serem infinitos, isto indica a existência de uma quantidade muito maior (não-enumerável) de linguagens não-gramaticais do que de linguagens formalizáveis através de gramáticas (enumerável), conforme pode ser inferido a partir do que foi demonstrado no Teorema 1.7. ■

O Teorema 7.10 estabelece a existência de linguagens não-gramaticais. Um exemplo importante de linguagem que não é recursivamente enumerável, e portanto não-gramatical, é a linguagem:

$$L_N = \overline{L_U} = \{C(M)w \in \Sigma^* \mid w \notin L(M)\}$$

A linguagem L_D compreende o conjunto das cadeias $C(M)w$ tais que w representa uma cadeia que não é aceita pela Máquina de Turing M, e corresponde ao complemento da linguagem L_U introduzida no Teorema 6.4 e posteriormente discutida no Teorema 7.7. A demonstração dessa afirmação é feita no Teorema 7.12.

Teorema 7.12 (L_N não-gramatical) *A linguagem L_N é não-gramatical.*

Justificativa Suponha-se que L_N seja uma linguagem recursivamente enumerável. Então, deve necessariamente existir pelo menos uma Máquina de Turing M_N que aceite L_N. Portanto, $L(M_N) = L_N$.

Como certamente existe pelo menos uma Máquina de Turing que aceita pelo menos uma cadeia, então $L_N \neq \Sigma^*$ e, conseqüentemente, $L(M_N) \neq \Sigma^*$. Logo, é fato que existe pelo menos uma cadeia $w_N \notin L(M_N)$.

Por outro lado, a cadeia $C(M_N)w_N$ deve pertencer a L_N, por definição desta última.

Deve-se, agora, considerar as seguintes duas únicas possibilidades para a cadeia $C(M_N)w_N$, no que se refere a M_N:

1. $C(M_N)w_N \in L(M_N)$, ou
2. $C(M_N)w_N \notin L(M_N)$.

Se a condição (1) for verdadeira, então $C(M_N)w_N \notin L_N$, ou seja, $C(M_N)w_N \notin L(M_N)$, o que é uma contradição com (1). Se, por outro lado, a condição (2) for verdadeira, então $C(M_N)w_N \in L_N$, ou seja, $C(M_N)w_N \in L(M_N)$, o que também é uma contradição com (2).

Logo, a hipótese original não é válida e, portanto, L_N não pode ser recursivamente enumerável. ■

Outra linguagem clássica, que não é recursivamente enumerável, é a linguagem:

$$L_D = \overline{L_K} = \{w_i \mid w_i \notin M_i\}$$

Assim como a linguagem L_K discutida no Teorema 6.5, ela supõe uma certa ordenação (por exemplo, lexicográfica) das cadeias w_i sobre um certo alfabeto Σ, e também do conjunto das Máquinas de Turing sobre o mesmo alfabeto. Diferentemente de L_K, no entanto, ela compreende as cadeias w_i que não são aceitas pelas Máquinas de Turing M_i correspondentes.

Teorema 7.13 (L_D não-gramatical) *A linguagem L_D é não-gramatical.*

Justificativa Suponha-se que L_D seja uma linguagem recursivamente enumerável. Então, deve existir necessariamente pelo menos uma Máquina de Turing M_D que aceite L_D, ou seja, $L(M_D) = L_D$.

Considere-se a cadeia w_D correspondente. Uma das duas seguintes situações deve necessariamente ser verdadeira:

1. $w_D \in L(M_D)$, ou
2. $w_D \notin L(M_D)$.

Se a condição (1) for verdadeira, isso significa que $w_D \notin L_D$. Mas, como $L_D = L(M_D)$, isso implica $w_D \notin L(M_D)$, uma contradição com (1). Se a condição (2) for

verdadeira, por outro lado, então $w_D \in L_D$. Mas como $L_D = L(M_D)$, isso implica $w_D \in L(M_D)$, uma contradição com (2).

Como a contradição acontece em ambos os casos, conclui-se que a hipótese inicial é falsa e, portanto, que L_D não pode ser gramatical. ■

7.8 Propriedades de Fechamento

A seguir serão demonstradas algumas das propriedades de fechamento mais importantes das linguagens recursivamente enumeráveis. Esta classe de linguagens é fechada em relação às operações de:

- União
- Concatenação
- Fecho de Kleene
- Intersecção

As correspondentes demonstrações serão apresentadas a seguir. As linguagens recursivamente enumeráveis, no entanto, não são fechadas em relação à operação de:

- Complementação

resultado este que será demonstrado na seqüência.

Teorema 7.14 (Fecho na união) *As linguagens recursivamente enumeráveis são fechadas em relação à operação de união.*

Justificativa Sejam L_1 e L_2 duas linguagens recursivamente enumeráveis quaisquer. Então, L_1 é aceita por uma Máquina de Turing M_1 e L_2 é aceita por uma Máquina de Turing M_2. A linguagem $L_3 = L_1 \cup L_2$ é aceita por M_3, construída conforme o Algoritmo 7.8.

Algoritmo 7.8 (Fecho na união) *Construção de uma Máquina de Turing que aceita a união de duas linguagens recursivamente enumeráveis a partir das Máquinas de Turing que as aceitam.*

- Entrada: duas Máquinas de Turing M_1 e M_2;
- Saída: uma Máquina de Turing M_3 que aceita $L(M_1) \cup L(M_2)$;
- Método:

    1. M_3 possui duas fitas de entrada;
    2. Uma cadeia w é gravada na primeira fita;
    3. M_3 copia w da primeira para a segunda fita;
    4. M_3 simula M_1 e M_2 simultaneamente, utilizando ambas as fitas de entrada e executando, alternadamente, movimentos de M_1 e de M_2;

5. Se M_1 aceita w, então M_3 pára e aceita w;

6. Se M_2 aceita w, então M_3 pára e aceita w.

M_3 aceita w se e somente se $w \in L_1$ ou $w \in L_2$. Logo, L_3 é uma linguagem recursivamente enumerável. ■

Teorema 7.15 (Fecho na concatenação) *As linguagens recursivamente enumeráveis são fechadas em relação à operação de concatenação.*

Justificativa Sejam L_1 e L_2 duas linguagens recursivamente enumeráveis quaisquer. Então, L_1 é aceita por uma Máquina de Turing M_1 e L_2 é aceita por uma Máquina de Turing M_2. A linguagem $L_3 = L_1L_2$ é aceita por M_3, construída conforme o Algoritmo 7.9.

Algoritmo 7.9 (Fecho na concatenação) *Construção de uma Máquina de Turing que aceita a concatenação de duas linguagens recursivamente enumeráveis a partir das Máquinas de Turing que as aceitam.*

- Entrada: duas Máquinas de Turing M_1 e M_2;
- Saída: uma Máquina de Turing M_3 que aceita $L(M_1)L(M_2)$;
- Método:

  1. M_3 inicialmente simula M_1 com uma cadeia de entrada w, $w = \alpha\beta$;
  2. Se M_1 aceita um prefixo α de w, então M_3 simula M_2 com o sufixo β;
  3. Se M_2 aceita β, então M_3 pára e aceita a cadeia w.

M_3 aceita w se e somente se $\alpha \in L_1$ e $\beta \in L_2$. Logo, L_3 é uma linguagem recursivamente enumerável. ■

Teorema 7.16 (Fecho no fecho) *As linguagens recursivamente enumeráveis são fechadas em relação à operação de fecho de Kleene.*

Justificativa Seja L_1 uma linguagem recursivamente enumerável qualquer. Então, L_1 é gerada por uma gramática irrestrita $G_1 = (V_1, \Sigma, P_1, S_1)$. A gramática $G_2 = (V_2, \Sigma, P_2, S_2)$, apresentada a seguir, e construída a partir de G_1 conforme o Algoritmo 7.10, gera a linguagem L_1^*.

Algoritmo 7.10 (Fecho no fecho) *Construção de uma gramática irrestrita que gera o fecho de Kleene de uma linguagem definida através de uma gramática irrestrita.*

- Entrada: uma gramática irrestrita G_1;
- Saída: uma gramática irrestrita G_2 tal que $L(G_2) = L(G_1)^*$;

- Método:

    1. $V_2 \leftarrow V_1 \cup \{S_2\}$
    2. $P_2 \leftarrow P_1 \cup \{S_2 \rightarrow S_1 S_2, S_2 \rightarrow \epsilon\}$

Como G_2 também é uma gramática irrestrita, então $L(G_2) = L_1^*$ é recursivamente enumerável. ■

Teorema 7.17 (Fecho na intersecção) *As linguagens recursivamente enumeráveis são fechadas em relação à operação de intersecção.*

Justificativa Sejam L_1 e L_2 duas linguagens recursivamente enumeráveis quaisquer. Então, L_1 é aceita por uma Máquina de Turing M_1 e L_2 é aceita por uma Máquina de Turing M_2. A linguagem $L_3 = L_1 \cap L_2$ é aceita por M_3, construída conforme o Algoritmo 7.11.

Algoritmo 7.11 (Fecho na intersecção) *Construção de uma Máquina de Turing que aceita a intersecção de duas linguagens recursivamente enumeráveis a partir das Máquinas de Turing que as aceitam.*

- Entrada: duas Máquinas de Turing M_1 e M_2;
- Saída: uma Máquina de Turing M_3 que aceita $L(M_1) \cap L(M_2)$;
- Método:

    1. M_3 possui duas fitas de entrada;
    2. Uma cadeia w é gravada na primeira fita;
    3. M_3 copia w da primeira para a segunda fita;
    4. M_3 simula M_1, utilizando a primeira fita;
    5. Se M_1 aceita w, então M_3 simula M_2 com a segunda fita;
    6. Se M_2 aceita w, então M_3 pára e aceita w.

M_3 aceita w se e somente se $w \in L_1 \cap L_2$. Logo, L_3 é uma linguagem recursivamente enumerável. ■

Teorema 7.18 (Fecho na complementação) *As linguagens recursivamente enumeráveis não são fechadas em relação à operação de complementação.*

Justificativa É suficiente mostrar que existe pelo menos uma linguagem recursivamente enumerável cujo complemento não é recursivamente enumerável.

A linguagem $L_U = \{C(M)w \in \Sigma^* \mid w \in L(M)\}$, conforme demonstrado nos Teoremas 6.4 e 7.7, é um exemplo de linguagem recursivamente enumerável, não-recursiva. Seu

complemento, no entanto (a linguagem $L_N = \{C(M)w \in \Sigma^* \mid w \notin L(M)\}$), conforme demonstrado no Teorema 7.11, é uma linguagem não-gramatical.

Logo, a classe das linguagens recursivamente enumeráveis não é fechada em relação à operação de complementação. ■

Apesar de a classe das linguagens recursivamente enumeráveis não ser fechada em relação à operação de complementação, um outro resultado interessante e bastante útil na demonstração de certos teoremas, envolvendo tanto a classe das linguagens recursivas quanto a própria classe das linguagens recursivamente enumeráveis, será demonstrado a seguir.

Teorema 7.19 (Recursivamente enumeráveis e seu complemento) *Se uma linguagem e também seu complemento forem recursivamente enumeráveis, então ambas as linguagens, além de serem recursivamente enumeráveis, são também recursivas.*

Justificativa Sejam L_1 e $L_2 = \Sigma^* - L_1$ duas linguagens recursivamente enumeráveis sobre o alfabeto Σ. Então, existem Máquinas de Turing M_1 e M_2 que aceitam, respectivamente, L_1 e L_2. Constrói-se uma Máquina de Turing M_3 que decide L_1, conforme o Algoritmo 7.12.

Algoritmo 7.12 (Recursivamente enumeráveis e seu complemento) *Construção de uma Máquina de Turing que decide L_1 a partir das Máquinas de Turing que aceitam L_1 e seu complemento.*

- Entrada: duas Máquinas de Turing M_1 e M_2, tais que $L(M_2) = \overline{L(M_1)}$;
- Saída: uma Máquina de Turing M_3 que decide $L(M_1)$;
- Método:

    1. M_3 possui duas fitas de trabalho;
    2. Uma cadeia w é gravada na primeira fita;
    3. M_3 copia w da primeira para a segunda fita;
    4. M_3 simula M_1 e M_2 simultaneamente, utilizando ambas as fitas de trabalho e executando, alternadamente, movimentos em M_1 e M_2;
    5. Se M_1 aceita w, então M_3 pára e aceita w;
    6. Se M_2 aceita w, então M_3 pára e rejeita w.

Qualquer que seja a cadeia de entrada $w \in \Sigma^*$, é certo que, após um número finito de movimentações, ou M_1 ou M_2, de maneira exclusiva, aceitará w. Se w for aceita por M_1, então será aceita por M_3. Se w for aceita por M_2, então será rejeitada por M_3.

Logo, M_3 aceita w se e somente se $w \in L_1$, e rejeita w se e somente se $w \in L_2$, ou seja, se $w \notin L_1$. Logo, L_1 é uma linguagem recursiva. ■

7.9 Questões Decidíveis e Não-Decidíveis

Nenhum dos principais problemas de decisão mais comuns é decidível para a classe das linguagens recursivamente enumeráveis:

- $w \in L$?
- $L = \varnothing$?
- $L = \Sigma^*$?
- $L_1 = L_2$?
- $L_1 \subseteq L_2$?
- $L_1 \cap L_2 = \varnothing$?

A demonstração é feita provando-se que, em todos os casos, as linguagens que representam os respectivos problemas não são recursivas. As demonstrações não serão apresentadas neste texto, mas podem ser encontradas em [46].

8 Conclusões

O estudo das classes de linguagens discutidas neste livro, apresentadas em ordem crescente de complexidade, e configuradas como conjuntos sucessivamente mais abrangentes, permite que se obtenha uma visão estruturada do assunto, com especial ênfase nos aspectos que diferenciam cada classe estudada da respectiva classe antecessora.

O presente capítulo apresenta um resumo dos principais resultados registrados nos capítulos anteriores, oferecendo, assim, uma visão panorâmica da essência contida em cada um desses capítulos, e também dos itens que fazem com que certas classes de linguagens se assemelhem ou se diferenciem.

Por outro lado, a aplicabilidade de linguagens formais e autômatos, nos campos prático e teórico, não se encerra em si mesma, apresentando grande relevância para áreas que serão destacadas mais adiante neste capítulo.

8.1 Uma Hierarquia de Classes de Linguagens

Este texto apresentou os principais resultados da teoria de linguagens formais e autômatos, seguindo estritamente uma seqüência que se iniciou com a classe de linguagens mais restrita conhecida (a das linguagens regulares), e se encerrou com a classe de linguagens mais ampla conhecida (a das linguagens recursivamente enumeráveis), passando gradativamente de uma classe mais restrita para a mais geral seguinte, dentro de uma escala de complexidade crescente, baseada na Hierarquia de Chomsky, e procurando sempre ampliar o horizonte do leitor, sem antecipar modelos, conceitos ou teorias relativos às classes de linguagens mais avançadas.

A Figura 8.1 resume alguns dos principais resultados colhidos ao longo deste estudo,[1] para cada classe de linguagens considerada, e que serão brevemente revistos na seqüência.

[1] "N.A." indica "Não se aplica" e "?" indica que o resultado não é conhecido até a presente data.

Tipo (Hierarquia de Chomsky)	Classe de linguagens	Gramática	Reconhecedor	Reconhecedor determinístico ≡ não-determinístico?	Estruturas sintáticas típicas da classe de linguagens
3	Regular	Regular	Autômato finito	Sim	Repetição, união e concatenação de termos
2	Livre de contexto determinística descendente	LL(k)	Autômato de pilha determinístico	N.A.	Aninhamento de construções sintáticas
	Livre de contexto determinística ascendente	LR (k)	Autômato de pilha determinístico	N.A.	?
	Livre de contexto não-ambígua	Livre de contexto não-ambígua	Autômato de pilha	Não	?
	Livre de contexto	Livre de Contexto	Autômato de pilha	Não	?
1	Sensível ao contexto	Sensível ao contexto	Máquina de Turing com fita limitada	?	Dependência entre termos
0	Recursiva	?	Máquina de Turing que sempre pára	Sim	?
	Recursiva-mente enumerável	Irrestrita	Máquina de Turing	Sim	?
N.A	Não-gramaticais	N.A.	?	N.A.	?

Figura 8.1: Classes de linguagens e suas características principais

A partir da Hierarquia de Chomsky original (definida em [61] e [62]), foram identificadas, caracterizadas e situadas importantes classes adicionais de linguagens: a das linguagens livres de contexto determinísticas, a das linguagens livres de contexto não-ambíguas, a das linguagens recursivas e a das linguagens não-gramaticais.

Excluindo, naturalmente, a classe das linguagens não-gramaticais, todas as demais classes podem ser caracterizadas em termos gramaticais — exceto a classe das linguagens recursivas, cuja definição é baseada unicamente no conceito de uma Máquina de Turing que sempre pára, e não em gramáticas.

Do ponto de vista do reconhecimento, três modelos distintos, com seis variantes importantes, foram apresentados: os autômatos finitos, os autômatos de pilha (nas versões determinística e não-determinística) e as Máquinas de Turing (nas versões com fita limitada, sem limitação de fita e que sempre param, e sem limitação de fita, que podem ou não parar). Cada um desses seis tipos de dispositivo está associado ao reconhecimento de uma classe de linguagens própria.

Quando a questão da equivalência entre as versões determinística e não-determinística de um mesmo modelo de reconhecedor é considerada, sabe-se com certeza que apenas os autômatos de pilha possuem versões que aceitam classes diferentes de linguagens. Exceto a Máquina de Turing com fita limitada, para a qual ainda não se conseguiu

demonstrar até hoje se existe ou não equivalência entre as versões determinística e não-determinística ([58]), todos os demais modelos e tipos de reconhecedor possuem versões que são comprovadamente equivalentes quanto a esse quesito.

Finalmente, é conveniente considerar as diferentes classes de linguagens do ponto de vista exclusivo das propriedades sintáticas que as distingüem umas das outras — independentemente do tipo de gramáticas ou reconhecedores empregados para a sua especificação. Dessa forma, pelo menos três classes de linguagens podem ser facilmente identificadas: a das linguagens regulares, a das linguagens livres de contexto determinísticas e a das linguagens sensíveis ao contexto. As demais classes de linguagens, até o ponto em que se sabe, não exibem propriedades sintáticas conhecidas que se possa considerar como "típicas" de todas as linguagens da mesma classe.

Conforme resultados demonstrados ao longo do texto, todas as classes de linguagens consideradas constituem uma hierarquia de inclusão própria, que começa com a classe das linguagens regulares e termina com a classe das linguagens não-gramaticais. A Figura 8.2 ilustra a referida hierarquia, incluindo exemplos de linguagens que tornam a relação de inclusão própria em cada nível.

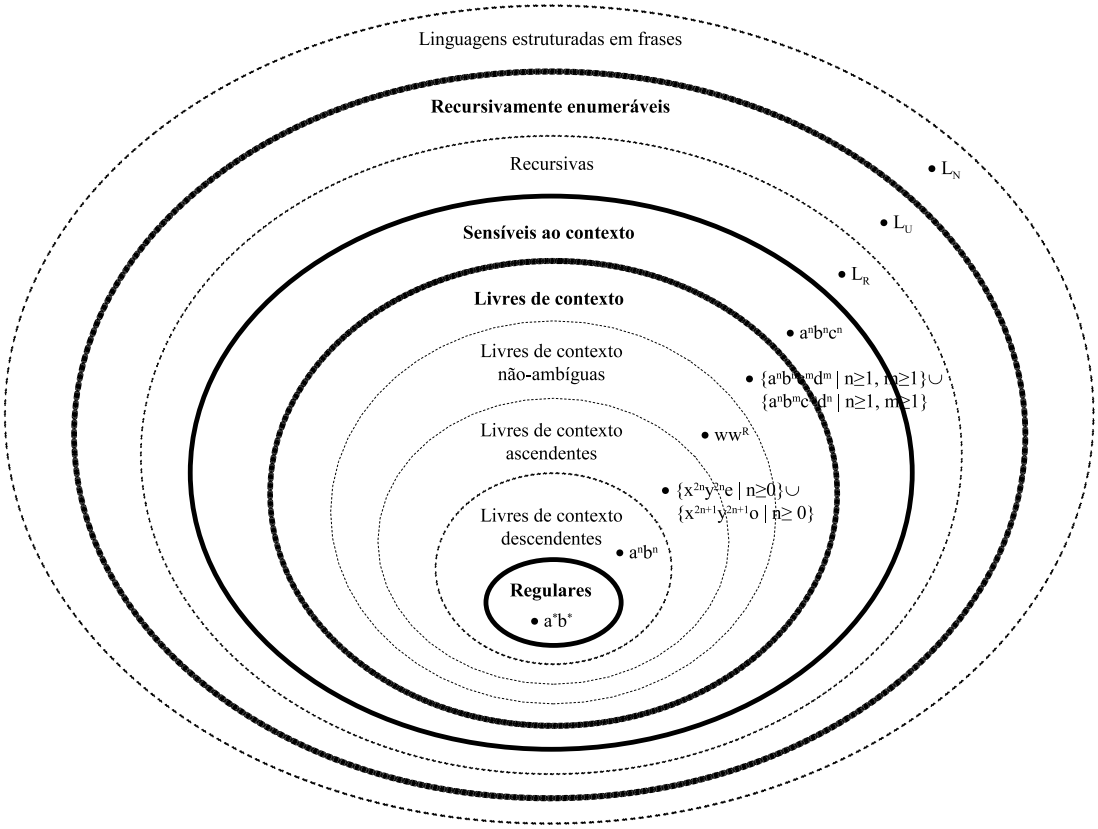

Figura 8.2: Hierarquia de inclusão própria das classes de linguagens, com exemplos

As Figuras 8.3 e 8.4 apresentam um resumo das principais propriedades de fechamento e das principais questões de decisão consideradas para cada classe de linguagens. Onde empregada, a letra "N" indica que a classe não é fechada em relação a uma de-

terminada propriedade ou, ainda, que a questão não é decidível para aquela classe de linguagens.

	Regulares	Livres de contexto	Sensíveis ao contexto	Recursivas	Recursivamente enumeráveis
União	✓	✓	✓	✓	✓
Concatenação	✓	✓	✓	✓	✓
Intersecção	✓	N	✓	✓	✓
Complementação	✓	N	✓	✓	N
Fechamento	✓	✓	N	N	✓

Figura 8.3: Resumo das principais propriedades de fechamento

	Regulares	Livres de contexto	Sensíveis ao contexto	Recursivas	Recursivamente enumeráveis
$w \in L$?	✓	✓	✓	✓	N
$L = \varnothing$?	✓	✓	N	N	N
$L = \Sigma^*$?	✓	N	N	N	N
$L_1 = L_2$?	✓	N	N	N	N
$L_1 \subseteq L_2$?	✓	N	N	N	N
$L_1 \cap L_2 = \varnothing$?	✓	N	N	N	N

Figura 8.4: Resumo das principais propriedades de decisão

8.2 Decidibilidade e Complexidade

O estudo das linguagens formais e autômatos é fundamental para o entendimento e o domínio de várias outras áreas da computação, mas duas se destacam perante as demais:

- Construção de processadores para linguagens de programação;
- Estudo da complexidade de algoritmos.

As linguagens livres de contexto são a base para a especificação de linguagens de programação de alto nível, e também para o projeto e a implementação de seus respectivos analisadores sintáticos. Estes, por sua vez, são a base para a estruturação de compiladores e interpretadores para as inúmeras linguagens usadas no dia-a-dia do profissional e do leigo em computação, que as empregam extensivamente, de forma direta ou indireta, consciente ou inconsciente. Pode-se dizer, portanto, que essa área constitui uma importante aplicação prática da teoria de linguagens formais e autômatos.

Por outro lado, no campo teórico, provavelmente a maior aplicação da teoria de linguagens e autômatos recai sobre o estudo da complexidade de algoritmos. Conforme discutido nos Capítulos 6 e 7, a constatação da decibilidade de linguagens confere a certeza da existência de algoritmos que possam ser usados para resolver a classe de problemas representada por tais linguagens. A constatação da indecidibilidade de uma linguagem, por sua vez, garante a inexistência de uma solução geral para o problema, quando se consideram todas as suas possíveis instâncias.

A questão, no entanto, não se resume à mera existência ou inexistência de algoritmos para se resolver determinados problemas.

Além da **decidibilidade** (também referida como **computabilidade**), deseja-se estabelecer os parâmetros que indiquem o grau de dificuldade na busca de soluções para um certo problema. Assim, não basta um problema ser decidível. É importante também que o mesmo possa ser solucionado (ou decidido) através de algoritmos que apresentem parâmetros considerados aceitáveis de desempenho. Problemas decidíveis, porém cujas soluções extrapolam os limites do que se considera aceitável, definitivamente não apresentam interesse prático.

Assim, o estudo da complexidade de algoritmos se inicia onde o estudo da decidibilidade de problemas se encerra. Uma vez analisada a decidibilidade de um certo problema, há que se considerar os custos envolvidos na busca dessa solução. Em função dessa análise, é possível que se justifiquem os esforços de busca da mesma, ou, então, o seu abandono.

Os custos envolvidos no estudo da complexidade de algoritmos normalmente se referem aos custos de execução dos mesmos, na busca das soluções pretendidas. Desses, sobressaem-se dois parâmetros objetivos, facilmente mensuráveis e com relação direta nas realizações mecanizadas efetuadas através de dispositivos computacionais. Trata-se do tempo total de execução e do volume total de memória necessários para se chegar à solução do problema através da execução do algoritmo em questão.

É comum que se estabeleçam limites máximos para esses parâmetros, que, vinculados ao tamanho do problema que se deseja resolver — normalmente representado pelo volume de dados a ser processado, ou, mais freqüentemente, pelo tamanho da cadeia de entrada a ser processada —, configuram a fronteira daquilo que se entende como critérios de aceitação de desempenho na execução de algoritmos.

Dessa forma, considera-se o estudo da **complexidade no tempo** e o estudo da **complexidade no espaço** para, respectivamente, designar a forma como o tempo total de execução de um algoritmo e o volume total de memória exigido por esse algoritmo variam conforme o tamanho da entrada.

Reunidos sob a designação conjunta e genérica de **complexidade**, esses estudos são complementares e normalmente antecedem quaisquer decisões referentes à alocação de recursos para solucionar problemas considerados inicialmente apenas como decidíveis. O estudo da complexidade é, portanto, uma extensão natural do estudo da computabilidade. Para o leitor interessado em aprofundar seus conhecimentos nessa área, recomenda-se o estudo de [46], [51] ou ainda [48].

Referências Bibliográficas

Adaptatividade

[1] **Adapting to Babel: adaptivity & context-sensitivity in parsing**, *Q. T. Jackson*, Ibis Publishing, 2006

[2] **Adaptive automata for context-dependent languages**, *J. J. Neto*, ACM SIGPLAN Notices, v.29, n.9, 1994

[3] **Adaptive automata for syntax learning**, *J. J. Neto and M. K. Iwai*, XXIV Conferencia Latinoamericana de Informática, Quito — Ecuador, Centro Latinoamericano de Estudios en Informática, Pontificia Universidad Católica del Ecuador, tomo 1, 1998

[4] **Solving complex problems efficiently with adaptative automata**, *J. J. Neto*, CIAA 2000 — Fifth International Conference on Implementation and Application of Automata, London, Ontario, Canada, 2000

[5] **A stochastic musical composer based on adaptive algorithms**, *B. A. Basseto e J. J. Neto*, Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação, vol. 3, 1999

[6] **Generation and recognition of formal languages by modifiable grammars**, *B. Burshteyn*, ACM SIGPLAN Notices v.25, n.12, 1990

[7] **Dynamic parsers and evolving grammars**, *S. Cabasino, P. S. Paolucci and G. M. Todesco*, ACM SIGPLAN Notices, v.27, n.11, 1992

[8] **A survey of adaptable grammars**, *H. Christiansen*, ACM SIGPLAN Notices, v.25, n.11, 1990

[9] **Um ambiente de desenvolvimento de reconhecedores sintáticos baseados em autômatos adaptativos**, *J. C. D. Pereira e J. J. Neto*, II Brazilian Symposium on Programming Languages, Campinas, SP, Brazil, 1997

[10] **Self-modifying finite automata: an introduction**, *R. S. Rubinstein and J. N. Shutt*, Information Processing Letters, v.56, n.4, 1995

[11] **Contribuição à metodologia de construção de compiladores**, *J. J. Neto*, Tese de livre-docência, Escola Politécnica da USP, São Paulo, 1993

[12] **STAD — Uma ferramenta para representação e simulação de sistemas através de statecharts adaptativos**, *J. R. A. Júnior*, Tese de doutorado, Escola Politécnica da USP, São Paulo, 1995

[13] **Um formalismo gramatical adaptativo para linguagens dependentes de contexto**, *M. K. Iwai*, Tese de doutorado, Escola Politécnica da USP, São Paulo, 2000

[14] **Alguns aspectos de tratamento de dependências de contexto em linguagem natural empregando tecnologia adaptativa**, *M. de Moraes*, Tese de doutorado, Escola Politécnica da USP, São Paulo, 2006

[15] **Um estudo do processo de inferência de gramáticas regulares e livres de contexto baseado em modelos adaptativos**, *I. P. Matsuno*, Dissertação de mestrado, Escola Politécnica da USP, 2006

[16] **Um formalismo adaptativo com mecanismos de sincronização para aplicações concorrentes**, *J. M. N. Santos*, Dissertação de mestrado, Escola Politécnica da USP, São Paulo, 1997

[17] **Mapeamento de ambientes desconhecidos por robôs móveis utilizando autômatos adaptativos**, *M. A. A. Sousa*, Dissertação de mestrado, Escola Politécnica da USP, São Paulo, 2006

[18] **Um método para a construção de analisadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos**, *C. E. Menezes*, Dissertação de mestrado, Escola Politécnica da USP, São Paulo, 2000

[19] **Recursive adaptable grammar**, *J. N. Shutt*, M. S. thesis, Computer Science Department, Worcester Polytecnic Institute, Worcester, Massachusetts, 1993

Autômatos de pilha estruturados

[20] **Design of a separable transition diagram compiler**, *M. E. Conway*, Communications of the ACM, 6, 7, 1963

[21] **A programmer's view of automata**, *B. H. Barnes*, ACM Computing Surveys, 4, 2, 1972

[22] **A formalization of transition diagram systems**, *D. B. Lomet*, Journal of the ACM, 20, 2, 1973

[23] **Reconhecedores sintáticos — uma alternativa didática para uso em cursos de engenharia**, *J. J. Neto e M. E. S. Magalhães*, Anais do XIV CNPD, São Paulo, 1981

Compiladores

[24] **Introdução à compilação**, *J. J. Neto*, Editora LTC, 1987

[25] **Compiler construction: an advanced course**, Editado por *F. L. Bauer* e *J. Eickel*, Springer-Verlag, 1976 - Segunda edição

[26] **The theory and practice of compiler writing**, *J. P. Tremblay and P. G. Sorenson*, McGraw-Hill, 1985

[27] **Crafting a compiler**, *C. N. Fischer and R. J. LeBlanc, Jr.*, Benjamin/Cummings, 1988

[28] **Compilers: Principles, techniques and tools**, *A. V. Aho, R. Sethi and J. D. Ullman*, Addison-Wesley, 1986; Segunda edição: *A. V. Aho, M. S. Lam, R. Sethi and J. D. Ullman*, 2007

[29] **Programming language processors in Java: Compilers and interpreters**, *D. A. Watt and D. F. Brown*, Prentice-Hall, 2000

Formalização de linguagens de programação

[30] **Semantics of programming languages**, *C. A. Gunter*, Massachusetts Institute of Technology, 1992

[31] **The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference**, *J. W. Backus*, Proc. of the International Conference on Information Processing, UNESCO, 1959

[32] **Report on the algorithmic language Algol 60**, *P. Naur*, Communications of the ACM 3:5, 1960

[33] **Revised report on the algorithmic language Algol 60**, *P. Naur*, Communications of the ACM 6:1, 1963

[34] **Revised report on the Algorithmic Language Algol 68**, *A. V. Wijngaarden*, Acta Informatica, v.5, n.1-3, 1975

[35] **What can we do about the unnecessary diversity of notation for syntactic definitions?**, *N. Wirth*, CACM, Vol. 20, issue 11, November 1977, pp. 822-823

[36] **Extended BNF**, *International Standards Organization*, ISO/IEC 14977, Primeira edição, 1996

Gramáticas com derivações controladas

[37] **Formal languages**, *A. Salomaa*, Academic Press, New York, 1973

[38] **Grammars with regulated rewriting**, *J. Dassow*, Em **Formal languages and applications**, *C. Martin-Vide, V. Mitrana e Gh. Paun*, Studies in Fuzziness and Soft Computing 148, Springer-Verlag, Berlin, 2004

[39] **Regulated rewriting in formal language theory**, *J. Dassow, Gh. Paun*, Akademie-Verlag, Berlin e Springer-Verlag, Berlin, 1989

[40] **Grammars with controlled derivations**, *J. Dassow, Gh. Paun, A. Salomaa*. Em **Handbook of formal languages I – III**, *G. Rozenberg, A. Salomaa*, Volume II, Chapter 3, Springer-Verlag, 1997

[41] **Gramáticas livres de contexto adaptativas com verificação de aparência**, *C. A. B. Pariente*, Tese de doutorado, Escola Politécnica da USP, 2004

[42] **Some problems of finite representability**, *E. Altman, R. Banerji*, Information and Control, 8, 1965

[43] **Some questions of language theory**, *S. Abraham*, International Conference on Computational Linguistics, 1965

[44] **Programmed grammars and classes of formal languages**, *D. J. Rosenkrantz*, Journal of the Association for Computing Machinery, 16, 1969

[45] **Periodically time-variant context-free grammars**, *A. Salomaa*, Information and Control, 17, 1970

Linguagens formais e autômatos & Teoria da computação

[46] **Introduction to automata theory, languages and computation**, *J. E. Hopcroft and J. D. Ullman*, Addison Wesley, 1979; Segunda edição: *J. E. Hopcroft, R. Motwani and J. D. Ullman*, 2001; Terceira edição: 2007

[47] **Formal languages and their relation to automata**, *J. E. Hopcroft and J. D. Ullman*, Addison Wesley, 1969

[48] **Languages and machines**, *T. A. Sudkamp*, Addison Wesley, 1988; Terceira edição: 2006

[49] **The theory of parsing, translation and compiling, volume 1**, *A. V. Aho and J. D. Ullman*, Prentice-Hall, 1972

[50] **Introduction to formal language theory** , *M. A. Harrison*, Addison Wesley, 1978

[51] **Elements of the theory of computation**, *H. R. Lewis and C. H. Papadimitriou*, Prentice-Hall, 1972; Segunda edição: 1998

[52] **Introduction to languages and the theory of computation**, *J. C. Martin*, McGraw-Hill, 1991

[53] **Theory of finite automata**, *J. Carroll and D. Long*, Prentice-Hall, 1989

[54] **Mathematical foundations of programming**, *F. S. Beckman*, Addison Wesley, 1980

[55] **Introduction to formal languages**, *G. E. Révész*, McGraw-Hill, 1983

[56] **Discrete structures, logic and computability**, *J. L. Hein*, Jones and Bartlett, 2002

[57] **Contribution to the founding of the theory of transfinite numbers**, *G. Cantor*, Dover Publications, 1955

[58] **An introduction to formal languages and automata**, *P. Linz*, Jones and Bartlett, 2001; Quarta edição: 2006

[59] **Handbook of theoretical computer science, vol.B: Formal models and semantics**, *J. van Leeuwen*, Elsevier Science Publishers, 1990

[60] **Representation of events in nerve nets and finite automata**, *S. C. Kleene*, Automata Studies, Princeton University Press, 1956

[61] **Three models for the description of language**, *N. Chomsky*, IRE Transactions on Information Theory 2, 1956

[62] **On certain formal properties of grammar**, *N. Chomsky*, Information and Control 2, 1959

[63] **A new normal form theorem for context-free phrase structure grammars**, *S. A. Greibach*, Journal of the ACM, 12:1

[64] **Nondeterministic space is closed under complementation**, *N. Immerman*, Journal of Computing 17, No.5, 1988

[65] **Classes of languages and linear bounded automata**, *S. Y. Kuroda*, Information and Control 7:2, 1964

Modelagem e UML

Ruby

[66] **Programming Ruby**, *D. Thomas*, Pragmatic Bookshelf, 2005