

# Linguagens Livres-do-Contexto

①

As Gramáticas livres-do-contesto são um método mais poderoso de descrever linguagens. Tais gramáticas podem descrever certas características que têm uma estrutura recursiva, o que as torna úteis em uma variedade de aplicações.

Foram utilizadas primeiramente no estudo de linguagens humanas.

Uma aplicação importante de gramáticas livres-do-contesto ocorre na especificação e compilação de linguagens de programação.

↳ Sintaxe de um linguagem de programação se torna uma referência para se aprender a linguagem.

↳ Projetistas de compiladores e interpretadores para LP freqüentemente começam obtendo uma gramática para a linguagem.

Compiladores têm um analisador sintático que entra e verifica se o código fonte está correto de acordo com as regras gramaticais/regras sintáticas da linguagem.

Metodologias facilitam a construção de analisadores sintáticos uma vez que é uma gramática livre-do-contesto esteja disponível. Algumas ferramentas até geram automaticamente esses analisadores. Yacc/bison para C/C++ no Linux.

## PLY para o python.

A coleção de linguagens associadas com gramáticas livres-do-contexto são denominadas linguagens livres-do-contexto. Incluem todas as linguagens regulares e muitas linguagens adicionais.

Linguagens livres-do-contexto são reconhecidas por Autômatos com Pilha. Esses autômatos são úteis porque nos permitem ganhar melhor percepção sobre o poder de linguagens livres-do-contexto.

# 2

## LINGUAGENS LIVRES-DO-CONTEXTO

No Capítulo 1 introduzimos dois métodos diferentes, embora equivalentes, de descrever linguagens: *autômatos finitos* e *expressões regulares*. Mostramos que muitas linguagens podem ser descritas dessa maneira, mas que algumas linguagens simples, como  $\{0^n 1^n \mid n \geq 0\}$ , não podem.

Neste capítulo, apresentamos *gramáticas livres-do-contexito*, um método mais poderoso de descrever linguagens. Tais gramáticas podem descrever certas características que têm uma estrutura recursiva, o que as torna úteis em uma variedade de aplicações.

Gramáticas livres-do-contexto foram primeiramente utilizadas no estudo de linguagens humanas. Uma maneira de entender o relacionamento de termos tais como *nome*, *verbos* e *preposição* e suas respectivas frases leva a uma referência natural, porque frases nominais podem aparecer dentro de frases verbais e vice-versa. Gramáticas livres-do-contexto podem capturar aspectos importantes desses relacionamentos.

Uma aplicação importante de gramáticas livres-do-contexto ocorre na especificação e compilação de linguagens de programação. Uma gramática para uma linguagem de programação freqüentemente aparece como uma referência para pessoas tentando aprender a sintaxe da linguagem. Projetistas de compiladores e interpretadores para linguagens de programação freqüentemente começam obtendo uma gramática para a linguagem. A maioria dos compiladores e interpretadores contém um componente chamado *analisador* que extrai o significado de um programa antes de gerar o código compilado ou realizar a

execução interpretada. Várias metodologias facilitam a construção de um analisador uma vez que uma gramática livre-do-contexro esteja disponível. Algumas ferramentas até geram automaticamente o analisador a partir da gramática.

A coleção de linguagens associadas com gramáticas livres-do-contexro são denominadas **linguagens livres-do-contexro**. Elas incluem todas as linguagens regulares e muitas linguagens adicionais. Neste capítulo, damos uma definição formal de gramáticas livres-do-contexro e estudamos as propriedades de linguagens livres-do-contexro. Também introduzimos **autômatos com pilha**, uma classe de máquinas que reconhecem as linguagens livres-do-contexro. Autômatos com pilha são úteis porque nos permitem ganhar melhor percepção sobre o poder de linguagens livres-do-contexro.

## 2.1 GRAMÁTICAS LIVRES-DO-CONTEXTO

Segue um exemplo de uma gramática livre-do-contexro, que chamamos  $G_1$ .

$$\begin{aligned} A &\rightarrow 0A1 \\ A &\rightarrow B \\ B &\rightarrow \# \end{aligned}$$

Uma gramática consiste de uma coleção de **regras de substituição**, também denominadas **produções**. Cada regra aparece como uma linha na gramática, compreendendo um símbolo e uma cadeia separados por uma seta. O símbolo é chamado de uma **variável**. A cadeia é constituída de variáveis e outros símbolos chamados de **terminais**. Os símbolos de variáveis frequentemente são representados por letras maiúsculas. Os terminais são análogos ao alfabeto de entrada e frequentemente são representados por letras minúsculas, números ou símbolos especiais. Uma variável é designada como a **variável inicial**. Ela geralmente ocorre no lado esquerdo da primeira regra. Por exemplo, a gramática  $G_1$  contém três regras. As variáveis de  $G_1$  são  $A$  e  $B$ , e  $A$  é a variável inicial. Seus terminais são  $0, 1, \text{ e } \#$ .

Você usa uma gramática para descrever uma linguagem gerando cada cadeia dessa linguagem da seguinte maneira.

1. Escreva a variável inicial. Ela é a variável no lado esquerdo da primeira regra, a menos que especificado em contrário.
2. Encontre uma variável que esteja escrita e uma regra que comece com essa variável. Substitua a variável escrita pelo lado direito dessa regra.
3. Repita o passo 2 até que não reste nenhuma variável.

$000\#111$  na gramática  $G_1$  é

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

Você também pode representar a mesma informação pictorialmente com uma **árvore sintática**. Um exemplo de árvore sintática está mostrado na Figura 2.1.

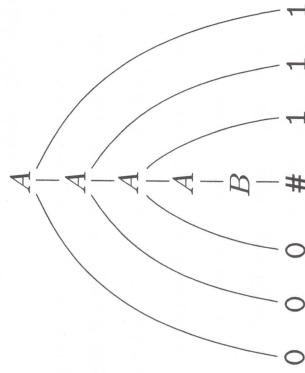


FIGURA 2.1  
Árvore sintática para  $000\#111$  na gramática  $G_1$ .

O conjunto de todas as cadeias geradas dessa maneira constitui a **linguagem da gramática**. Escrevemos  $L(G_1)$  para a linguagem da gramática  $G_1$ . Alguma experimentação com a gramática  $G_1$  nos mostra que  $L(G_1) = \{0^n\#\bar{1}^n \mid n \geq 0\}$ . Qualquer linguagem que pode ser gerada por alguma gramática livre-do-contexro é chamada uma **linguagem livre-do-contexro** (LLC). Por conveniência quando apresentamos uma gramática livre-do-contexro, abreviamos várias regras com a mesma variável no lado esquerdo, talis como  $A \rightarrow 0A1$  e  $A \rightarrow B$ , em uma única linha  $A \rightarrow 0A1 \mid B$ , usando o símbolo “|” como “ou”. O que vem a seguir é um segundo exemplo de gramática livre-do-contexro, chamada  $G_2$ , que descreve um fragmento da língua inglesa.

$$\begin{aligned} \langle \text{SENTENCE} \rangle &\rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\ \langle \text{NOUN-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\ \langle \text{VERB-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\ \langle \text{PREP-PHRASE} \rangle &\rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\ \langle \text{CMPLX-NOUN} \rangle &\rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\ \langle \text{CMPLX-VERB} \rangle &\rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\ \langle \text{ARTICLE} \rangle &\rightarrow \text{a} \mid \text{the} \\ \langle \text{NOUN} \rangle &\rightarrow \text{boy} \mid \text{girl} \mid \text{flower} \\ \langle \text{VERB} \rangle &\rightarrow \text{touches} \mid \text{likes} \mid \text{sees} \\ \langle \text{PREP} \rangle &\rightarrow \text{with} \end{aligned}$$

A gramática  $G_2$  tem dez variáveis (os termos com asterisco são ignorados).

ractere de espaço em branco); e 18 regras. As cadeias em  $L(G_2)$  incluem os três exemplos seguintes.

a boy sees  
the boy sees a flower  
a girl with a flower likes the boy

Cada uma dessas cadeias tem uma derivação na gramática  $G_2$ . A seguir está uma derivação da primeira cadeia nessa lista.

$$\begin{aligned} \langle \text{SENTENCE} \rangle &\Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\ &\Rightarrow \langle \text{CMPLX-NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\ &\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\ &\Rightarrow a \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\ &\Rightarrow a \text{ boy } \langle \text{VERB-PHRASE} \rangle \\ &\Rightarrow a \text{ boy } \langle \text{CMPLX-VERB} \rangle \\ &\Rightarrow a \text{ boy } \langle \text{VERB} \rangle \\ &\Rightarrow a \text{ boy sees} \end{aligned}$$

Na gramática  $G_1$ ,  $V = \{A, B\}$ ,  $\Sigma = \{0, 1, \#\}$ ,  $S = A$  e  $R$  é a coleção das três regras que aparecem na página 104. Na gramática  $G_2$ ,

$$\begin{aligned} V &= \{\langle \text{SENTENCE} \rangle, \langle \text{NOUN-PHRASE} \rangle, \langle \text{VERB-PHRASE} \rangle, \\ &\quad \langle \text{PREP-PHRASE} \rangle, \langle \text{CMPLX-NOUN} \rangle, \langle \text{CMPLX-VERB} \rangle, \\ &\quad \langle \text{ARTICLE} \rangle, \langle \text{NOUN} \rangle, \langle \text{VERB} \rangle, \langle \text{PREP} \rangle\}, \end{aligned}$$

$\Sigma = \{a, b, c, \dots, z, “ ”\}$ . O símbolo “ ” é o símbolo para o espaço em branco, colocado invisivelmente após cada palavra (a, boy etc.), de modo que as palavras não vão se juntar.

Freqüentemente especificamos uma gramática escrevendo somente suas regras. Podemos identificar as variáveis como os símbolos que aparecem no lado esquerdo das regras e os terminais como os símbolos remanescentes. Por convenção, a variável inicial é a colocada no lado esquerdo da primeira regra.

## EXEMPLOS DE GRAMÁTICAS LIVRES-DO-CONTEXTO

### EXEMPLO 2.3

Considere a gramática  $G_3 = (\{S\}, \{a, b\}, R, S)$ . O conjunto de regras,  $R$ , é

$$S \rightarrow aSb \mid SS \mid \varepsilon.$$

Essa gramática gera cadeias tais como abab, aaabb, e aababb. Você pode ver mais facilmente o que essa linguagem é se pensar em a como um parêntese à esquerda “ ” e b como um parêntese à direita “ ”. Vista dessa forma,  $L(G_3)$  é a linguagem de todas as cadeias de parênteses apropriadamente aninhados. ■

### EXEMPLO 2.4

Considere a gramática  $G_4 = (V, \Sigma, R, \langle \text{EXPR} \rangle)$ .  $V$  é  $\{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACTOR} \rangle\}$  e  $\Sigma$  é  $\{a, +, \times, (, )\}$ . As regras são

$$\begin{aligned} \langle \text{EXPR} \rangle &\rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle \\ \langle \text{TERM} \rangle &\rightarrow \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle \\ \langle \text{FACTOR} \rangle &\rightarrow \langle \text{EXPR} \rangle \mid a \end{aligned}$$

As duas cadeias  $a+a$  e  $(a+a)$  podem ser geradas com a gramática  $G_4$ . As árvores sintáticas são mostradas na Figura 2.5.

Um compilador traduz o código escrito em uma linguagem de programação para outra forma, usualmente mais adequada para a execução. Para fazer isso, o compilador extrai o significado do código a ser compilado em um processo chamado de **análise sintática**. Uma representação desse significado é a árvore sintática para o código na gramática livre-do-contexito para a linguagem de

## DEFINIÇÃO FORMAL DE UMA GRAMÁTICA LIVRE-DO-CONTEXTO

Vamos formalizar nossa noção de gramática livre-do-contexto (GLC).

### DEFINIÇÃO 2.2

Uma **gramática livre-do-contexo** é uma 4-upla  $(V, \Sigma, R, S)$ , onde

1.  $V$  é um conjunto finito denominado **variáveis**,
2.  $\Sigma$  é um conjunto finito, disjunto de  $V$ , denominado **terminais**,
3.  $R$  é um conjunto finito de **regras**, com cada regra sendo uma variável e uma cadeia de variáveis e terminais,
4.  $S \in V$  é a variável inicial.

Se  $u, v$  e  $w$  são cadeias de variáveis e terminais, e  $A \rightarrow w$  é uma regra da gramática, dizemos que  $uAv$  **origina**  $uwv$ , escrito  $uAv \Rightarrow uwv$ . Digamos que  $u$  **deriva**  $v$ , escrito  $u \xrightarrow{*} v$ , se  $u = v$  ou se existe uma sequência  $u_1, u_2, \dots, u_k$  para  $k \geq 0$  tal que

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$$

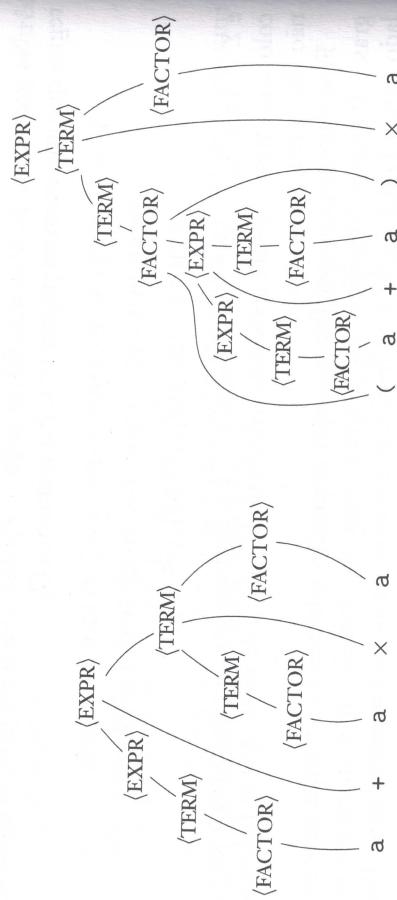


FIGURA 2.5

Árvores sintáticas para as cadeias  $a+a \times a$  e  $(a+a) \times a$ .

A gramática  $G_4$  descreve um fragmento de uma linguagem de programação que lida com expressões aritméticas. Observe como as árvores sintáticas na Figura 2.5 “agrupam” as operações. A árvore para  $a+a$  agrupa o operador  $\times$  e seus operandos (os dois últimos  $a$ ) como um operando do operador  $+$ . Na árvore para  $(a+a) \times a$ , o agrupamento é revertido. Esses agrupamentos estão de acordo com a precedência-padrão da multiplicação antes da adição e com o uso de parênteses para sobrepujar a precedência-padrão. A gramática  $G_4$  é projetada para capturar essas relações de precedência.

## PROJETANDO GRAMÁTICAS LIVRES-DO-CONTEXTO (GLC)

Tal qual com o projeto de autômatos finitos, discutido na Seção 1.1 (página 41), o projeto de gramáticas livres-do-conteúdo requer criatividade. De fato, gramáticas livres-do-conteúdo são ainda mais complicadas de construir que autômatos finitos porque estamos mais acostumados a programar uma máquina para tarefas específicas que a descrever linguagens com gramáticas. As técnicas seguintes são úteis, isoladamente ou em combinação, quando você se confronta com o problema de construir uma GLC.

Primeiro, muitas LLCs são a união de LLCs mais simples. Se você tem que construir uma GLC para uma LLC que você pode quebrar em partes mais simples, faça isso e ai então construa gramáticas individuais para cada parte. Essas gramáticas individuais podem ser facilmente reunidas em uma gramática para a linguagem original combinando suas regras e então adicionando a nova regra  $S \rightarrow S_1 | S_2 | \dots | S_k$ , onde as variáveis  $S_i$  são as variáveis iniciais para

Por exemplo, para obter uma gramática para a linguagem  $\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$ , primeiro construa a gramática

$$S_1 \rightarrow 0S_1 1 \mid \epsilon$$

para a linguagem  $\{0^n 1^n \mid n \geq 0\}$  e a gramática

$$S_2 \rightarrow 1S_2 0 \mid \epsilon$$

para a linguagem  $\{1^n 0^n \mid n \geq 0\}$  e então adicione a regra  $S \rightarrow S_1 | S_2$  para dar a

$$S \rightarrow S_1 | S_2$$

$$S_1 \rightarrow 0S_1 1 \mid \epsilon$$

$$S_2 \rightarrow 1S_2 0 \mid \epsilon$$

Segundo, construir uma GLC para uma linguagem que acontece de ser regular é difícil se você puder primeiramente construir um AFD para essa linguagem. Você pode converter qualquer AFD numa GLC equivalente da seguinte maneira. Pegue uma variável  $R_i$  para cada estado  $q_i$  do AFD. Adicione a regra  $R_i \rightarrow aR_j$  à GLC se  $\delta(q_i, a) = q_j$  for uma transição no AFD. Adicione a regra  $R_i \rightarrow \epsilon$  se  $q_i$  for estado de aceitação do AFD. Faça  $R_0$  a variável inicial da gramática, onde  $q_0$  é o estado inicial da máquina. Verifique, por você mesmo, que a GLC resultante gera a mesma linguagem que o AFD reconhece.

Terceiro, certas linguagens livres-do-conteúdo contêm cadeias com duas subcadeias que são “ligadas” no sentido de que uma máquina para uma linguagem como essa precisaria memorizar uma quantidade ilimitada de informação sobre uma das subcadeias para verificar que ela corresponde apropriadamente à outra subcadeia. Essa situação ocorre na linguagem  $\{0^n 1^n \mid n \geq 0\}$ , porque uma máquina precisaria memorizar o número de 0s de modo a verificar que ele é igual ao número de 1s. Você pode construir uma GLC para lidar com essa situação usando uma regra da forma  $R \rightarrow uRv$ , que gera cadeias nas quais a parte contendo os  $u$ s corresponde à parte contendo os  $v$ s.

Finalmente, em linguagens mais complexas, as cadeias podem conter certas estruturas que aparecem recursivamente como parte de outras estruturas (ou delas mesmas). Essa situação ocorre na gramática que gera expressões aritméticas no Exemplo 2.4. Sempre que o símbolo  $a$  aparece, em vez dele uma expressão parentalizada inteira pode aparecer recursivamente. Para atingir esse efeito, coloque o símbolo da variável que gera a estrutura na posição das regras correspondente à onde aquela estrutura pode aparecer recursivamente.

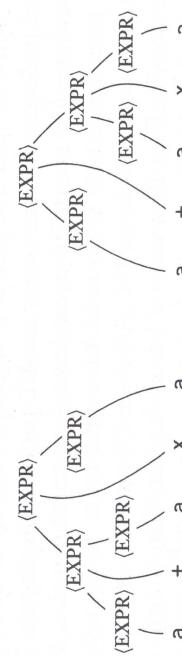
## AMBIGÜIDADE

As vezes uma gramática pode gerar a mesma cadeia de várias maneiras diferentes. Tal cadeia terá várias árvores sintáticas diferentes e, portanto, vários significados diferentes. Esse resultado pode ser indesejável para certas aplicações, como em linguagens de programação, onde um dado programa deve ter uma única interpretação.

Se uma gramática gera a mesma cadeia de várias maneiras diferentes, dizemos que a cadeia é derivada *ambigamente* nessa gramática. Se uma gramática gera alguma cadeia ambigamente, dizemos que a gramática é *ambígua*. Por exemplo, considere a gramática  $G_5$ :

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle^+ \langle \text{EXPR} \rangle \mid \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \mid (\langle \text{EXPR} \rangle) \mid a$$

Essa gramática gera a cadeia  $a + a \times a$  ambigamente. A figura abaixo mostra as duas árvores sintáticas diferentes.



**FIGURA 2.6**

As duas árvores sintáticas para a cadeia  $a + a \times a$  na gramática  $G_5$ .

Essa gramática não captura as relações de precedência usuais e, portanto, pode agrupar o  $+$  antes do  $\times$  ou vice-versa. Por outro lado, a gramática  $G_4$  gera exatamente a mesma linguagem, mas toda cadeia gerada tem uma árvore sintática única. Logo,  $G_4$  é não-ambígua, enquanto  $G_5$  é ambígua.

A gramática  $G_2$  (página 105) é um outro exemplo de uma gramática ambígua. A sentença "the girl touches the boy with the flower" tem duas derivações diferentes. No Exercício 2.8 pede-se que você dê as duas árvores sintáticas e observe sua correspondência com as duas maneiras diferentes de ler essa sentença.

Agora, formalizamos a noção de ambigüidade. Quando dizemos que uma gramática gera uma cadeia ambigamente, queremos dizer que a cadeia tem duas árvores sintáticas diferentes, e não duas derivações diferentes. Duas derivações podem diferir meramente pela ordem na qual elas substituem variáveis e assim não na sua estrutura geral. Para nos concentrarmos na estrutura, definimos

um tipo de derivação que substitui variáveis em uma ordem fixa. Uma derivação de uma cadeia  $w$  em uma gramática  $G$  é uma *derivação mais à esquerda* se a cada passo a variável remanescente mais à esquerda é aquela que é substituída. A derivação que precede a Definição 2.2 (página 106) é uma derivação mais à esquerda.

## DEFINIÇÃO 2.7

Uma cadeia  $w$  é derivada *ambigamente* na gramática livre-do-contesto  $G$  se ela tem duas ou mais derivações mais à esquerda diferentes. A gramática  $G$  é *ambígua* se ela gera alguma cadeia ambigamente.

As vezes quando temos uma gramática ambígua podemos encontrar uma gramática não-ambígua que gera a mesma linguagem. Algumas linguagens livres-do-contesto, entretanto, podem ser geradas apenas por gramáticas ambíguas. Tais linguagens são chamadas *inherentemente ambíguas*. O Problema 2.29 pede que você prove que a linguagem  $\{a^i b^j c^k \mid i = j \text{ ou } j = k\}$  é inherentemente ambígua.

## FORMA NORMAL DE CHOMSKY

Quando se trabalha com linguagens livres-do-contesto, é freqüentemente conveniente tê-las em forma simplificada. Uma das formas mais simples e mais úteis é chamada forma normal de Chomsky. A forma normal de Chomsky é útil quando se quer dar algoritmos para trabalhar com gramáticas livres-do-contesto, como fazemos nos Capítulos 4 e 7.

## DEFINIÇÃO 2.8

Uma gramática livre-do-contesto está na *forma normal de Chomsky* se toda regra é da forma

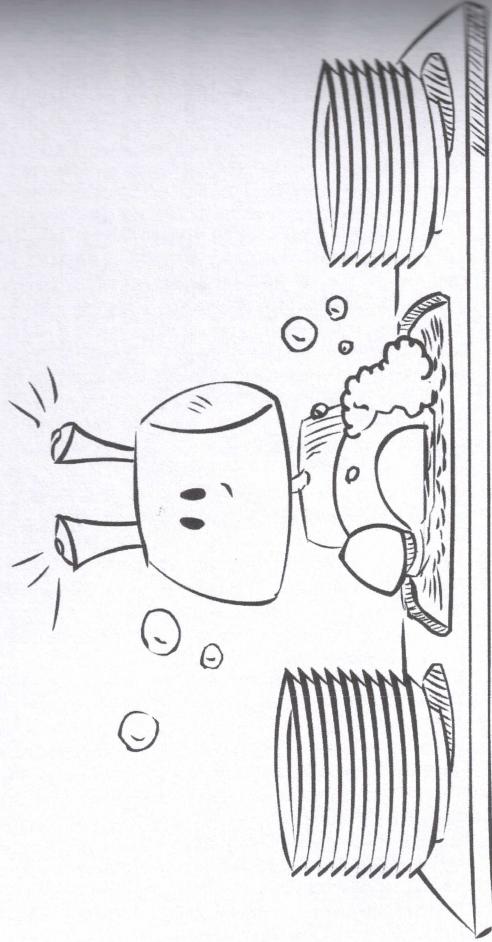
$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a \end{aligned}$$

onde  $a$  é qualquer terminal e  $A, B$  e  $C$  são quaisquer variáveis — exceto que  $B$  e  $C$  não podem ser a variável inicial. Adicionalmente, permitimos a regra  $S \rightarrow \epsilon$ , onde  $S$  é a variável inicial.

## capítulo

# linguagens livres do contexto

■ O estudo das linguagens livres do contexto é de fundamental importância, pois comprehende a sintaxe da maioria das linguagens de programação de propósitos gerais como Pascal, C e Java. Dois formalismos livres do contexto são apresentados: gramática livre do contexto e autômato com pilha. Alguns estudos e resultados são desenvolvidos objetivando reconhecimento de linguagens e prova de propriedades como árvores de derivação, simplificação de gramáticas e formas normais de gramáticas.



O estudo da *classe das linguagens livres do contexto ou tipo 2* é de fundamental importância na computação e informática pois:

- compreende um universo mais amplo de linguagens (comparativamente com o das regulares), tratando, adequadamente, questões como parênteses balanceados, construções bloco-restruturadas, entre outras, típicas de linguagens de programação como Pascal, C, Java, etc.;
- os algoritmos reconhecedores e geradores que implementam as linguagens livres do contexto são relativamente simples e possuem uma eficiência razoável;
- exemplos típicos de aplicações dos conceitos e resultados referentes às linguagens livres do contexto são centrados em *linguagens artificiais* e, em especial, nas linguagens de programação. Em particular, destacam-se analisadores sintáticos, tradutores de *linguagens de processadores de texto* em geral.

De acordo com a hierarquia de Chomsky, a classe das linguagens livres do contexto contém propriamente a classe das linguagens regulares. Entretanto, constitui uma classe de linguagens relativamente restrita, sendo fácil definir linguagens que não pertencem a esta classe.

- O estudo das *linguagens livres do contexto* é abordado, usando-se os seguintes formalismos:
- a** *Gramática livre do contexto*. Trata-se de um formalismo axiomático ou gerador o qual, como o nome indica, é uma gramática, mas com restrições na forma das regras de produção. Tais restrições são definidas de maneira mais livre que na gramática regular;
  - b** *Autômato com pilha*. Trata-se de um formalismo operacional ou reconhecedor cuja estrutura básica é análoga à do autômato finito não determinístico, adicionada de uma memória auxiliar tipo *pilha* (a qual pode ser lida ou gravada).

Relativamente às gramáticas livres do contexto, os seguintes tópicos são desenvolvidos e são importantes tanto para desenvolvimento e otimização de algoritmos reconhecedores, como na prova de teoremas:

- a** *Árvore de derivação*. Representa a derivação de uma palavra na forma de árvore, partindo do símbolo inicial como a *raiz*, e terminando em símbolos terminais como *folhas*;
- b** *Gramática ambígua*. Se existe pelo menos uma palavra que possua duas ou mais árvores de derivação nesta gramática;
- c** *Simplificação de gramática*. Simplifica as produções sem reduzir o poder de geração da gramática;
- d** *Forma normal*. Estabelece restrições rígidas na forma das produções, sem reduzir o poder de geração da gramática.

A construção de um autômato com pilha a partir de uma gramática livre do contexto qualquer, permite estabelecer as seguintes conclusões:

- a construção de um reconhecedor para uma linguagem livre do contexto a partir de sua gramática é simples e imediata;
- qualquer linguagem livre do contexto pode ser reconhecida por um autômato com pilha com somente um estado de controle lógico, o que implica que a estrutura de pilha é suficiente como única memória, não sendo necessário usar os estados para "memorizar" informações passadas.

## 6.1 → gramática livre do contexto

As *linguagens livres do contexto ou tipo 2* são definidas a partir das gramáticas *livres do contexto*.

**definição 6.1** – Gramática livre do contexto

Uma *gramática livre do contexto G* é uma gramática:

$$G = (V, T, P, S)$$

com a restrição de que qualquer regra de produção de *P* é da forma:  
 $A \rightarrow \alpha$

onde *A* é uma variável de *V*, e  $\alpha$ , uma palavra de  $(V \cup T)^*$ .  
 Portanto, uma gramática livre do contexto é uma gramática na qual o lado esquerdo das produções contém exatamente uma variável.

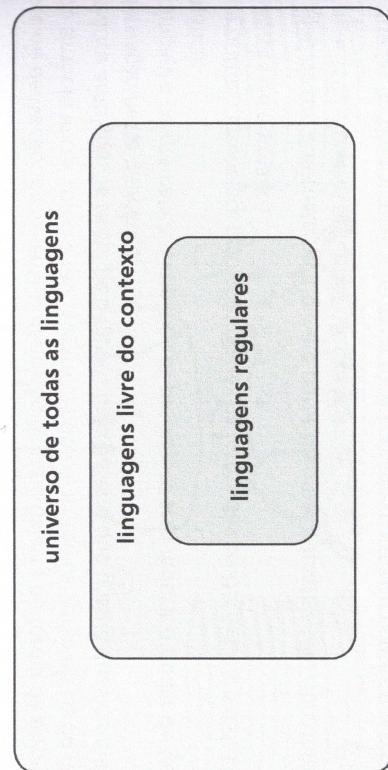
**definição 6.2** – Linguagem livre do contexto, linguagem tipo 2

Seja *G* uma gramática livre do contexto. A *linguagem gerada* pela gramática *G*:

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

é dita uma *linguagem livre do contexto ou linguagem tipo 2*.

Portanto, toda linguagem regular é livre do contexto. A relação entre as classes de linguagens estudadas até o momento é ilustrada na figura 6.1 (inclusões próprias).



**figura 6.1** Relação entre as classes de linguagens.

O nome “livre do contexto” deve-se ao fato de representar a mais geral classe de linguagens cuja produção é da forma  $A \rightarrow \alpha$ . Ou seja, em uma derivação, a variável  $A$  deriva  $\alpha$  sem depender (sendo “livre”) de qualquer análise dos símbolos que antecedem ou sucedem  $A$  (“contexto”) na palavra que está sendo derivada.

**exemplo 6.1** – Gramática livre do contexto: duplo balanceamento

Considere a linguagem:

$$L_1 = \{ a^n b^n \mid n \geq 0 \}$$

A seguinte gramática livre do contexto:

$$\begin{aligned} G_1 &= (\{ S \}, \{ a, b \}, P_1, S), \text{ em que:} \\ P_1 &= \{ S \rightarrow aSb \mid S \rightarrow \varepsilon \} \end{aligned}$$

é tal que  $GERA(G_1) = L_1$ . Por exemplo, a palavra  $aabb$  pode ser gerada pela seguinte sequência de derivação:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaebb = aabb$$

O exemplo acima é um caso clássico e de fundamental importância no estudo da computação e informática, pois permite estabelecer analogia com estruturas de duplo平衡amento em linguagens de programação como Pascal, C, Java, etc. Por exemplo:

- a** Linguagens bloco-estruturadas como `begin`<sup>n</sup> e similares;
- b** Linguagens com parênteses balanceados em expressões na forma  $(^n )^n$

O exemplo a seguir ilustra parênteses balanceados em expressões aritméticas.

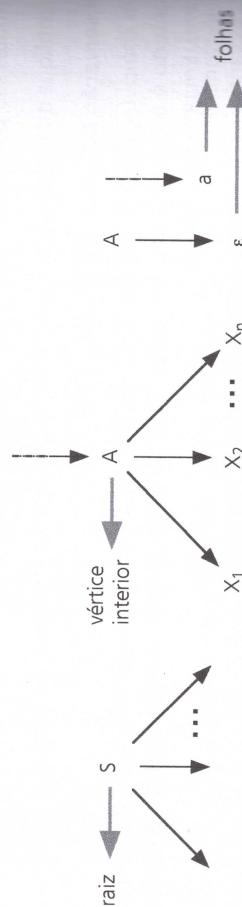
## 6.2 → árvore de derivação

- Em algumas aplicações como compiladores e processadores de textos, frequentemente é conveniente representar a derivação de palavras na forma de árvore, partindo do símbolo inicial como a *raiz*, e terminando em símbolos terminais como *folhas*.

**definição 6.4 – Árvore de derivação**

Para uma determinada gramática livre do contexto, a representação da derivação de palavra na forma de árvore, denominada árvore de derivação, é como segue (considere a figura 6.2):

- a** A raiz é o símbolo inicial da gramática;
- b** Os vértices interiores são os “filhos” de A, então:
  - $A \rightarrow X_1 X_2 \dots X_n$  é uma produção da gramática;
  - os vértices  $X_1, X_2, \dots, X_n$  são ordenados da esquerda para a direita;
  - Um vértice folha, ou simplesmente folha, é um símbolo terminal, ou o símbolo vazio ( $\epsilon$ ). Neste caso, o vazio é o único filho de seu pai ( $A \rightarrow \epsilon$ ).
- c** **figura 6.2** Árvore de derivação: representação.

**figura 6.2** Árvore de derivação: representação.**exemplo 6.4 – Árvore de derivação**

A palavra **aabb** e a expressão  $[x+x]*x$  dos seguintes exemplos, respectivamente:

- exemplo 6.1 – Gramática livre do contexto: duplo balanceamento;
- exemplo 6.2 – Gramática livre do contexto: expressões aritméticas;

são geradas pelas árvores de derivação ilustradas na figura 6.3, à esquerda e à direita, respectivamente.

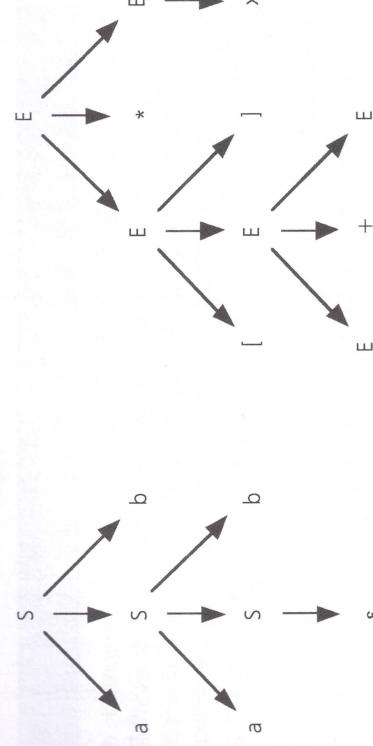
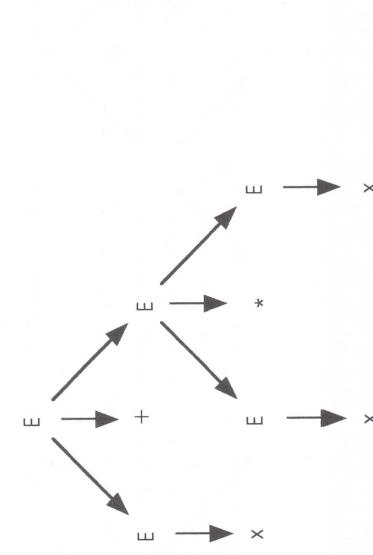
Uma única árvore de derivação pode representar derivações distintas de uma mesma palavra, como ilustrado no exemplo a seguir.

**exemplo 6.5 – Árvore de derivação × derivações**

Na árvore representada na figura 6.4, a palavra  $x+x*x$  pode ser gerada por diversas derivações distintas, como segue:

- a**  $E \Rightarrow E+E \Rightarrow x+E \Rightarrow x+E*E \Rightarrow x+x*E \Rightarrow x+x*x$
- b**  $E \Rightarrow E+E \Rightarrow E+E*x \Rightarrow E+E*E \Rightarrow x+E*E \Rightarrow x+x*x \Rightarrow x+x*x$
- c**  $E \Rightarrow E+E \Rightarrow E+E*x \Rightarrow x+E*E \Rightarrow x+x*x \Rightarrow x+x*x$
- d** etc...

Observe que, no item a), em cada passo de derivação, sempre é derivada a variável mais à esquerda. Analogamente para a variável mais à direita no item b).

**figura 6.2** Árvore de derivação: representação.**figura 6.3** Árvore de derivação: aabb e  $[x+x]*x$ .**definição 6.5 – Derivação mais à esquerda (direita)**

Dada uma árvore de derivação, uma derivação mais à esquerda (respectivamente, derivação mais à direita) de uma palavra é a sequência de produções aplicada sempre à variável mais à esquerda (respectivamente, mais à direita) da palavra, em cada passo de derivação.

**exemplo 6.6 – Derivação mais à esquerda (direita)**

Observe novamente o exemplo 6.5 – Árvore de derivação × derivações. Então:

- o item a) representa uma derivação mais à esquerda;
- o item b) representa uma derivação mais à direita.

6.3

→ gramática livre do contexto ambígua

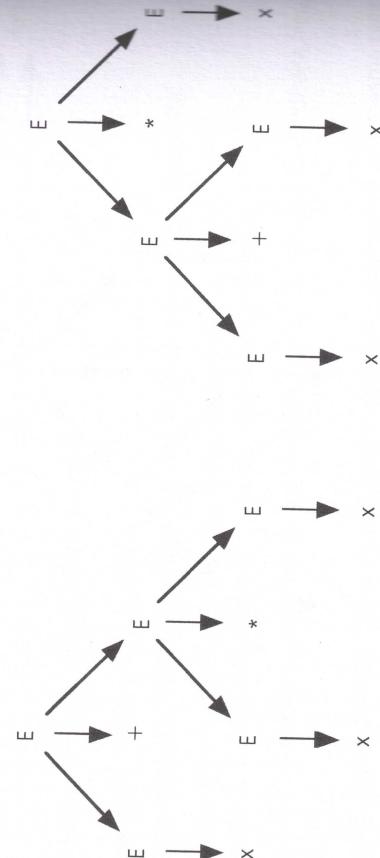
eventualmente, uma mesma palavra pode ser associada a duas ou mais árvores de derivação, caracterizando uma gramática *ambígua*. Em muitas aplicações como, por exemplo, no desenvolvimento e otimização de alguns tipos de algoritmos de reconhecimento, é conveniente que a gramática usada seja não ambígua. Entretanto, nem sempre é possível eliminar ambiguidades. Na realidade, é fácil definir linguagens para as quais qualquer gramática livre de contexto é ambígua.

## **Definição 6.6 – Gramática ambíqua**

ma gramática livre do contexto é dita uma gramática *livre do contexto* ou *ambigua* ou *lesmente gramática ambigua*, se existe pelo menos uma palavra que possua duas ou mais vóres de derivacão nessa gramática.

Example 67

considere o exemplo 6.2 – Gramática livre do contexto: expressões aritméticas. A palavra **XXX** pode ser gerada por árvores distintas, como ilustrado na figura 6.5. Portanto, a gramática em questão é ambígua.



**figura 6.5** Gramática ambígua: árvores diferentes para a palavra x+xx\*.

considere a gramática ambígua apresentada no exemplo 6.2 – Gramática livre do contexto expressões aritméticas. A palavra **x+x\*x** possui mais de uma derivação à esquerda (respectivamente,  $x + x \cdot x$  e  $x + (x \cdot x)$ ).

→ gramática livre do contexto ambíguo

Uma forma equivalente de definir gramática *ambígua* é verificando a existência de pelo menos uma palavra com duas ou mais derivações mais à esquerda (ou, alternativamente, mais à direita). O seguinte teorema não será demonstrado.

**Teorema 6.7** = Gramática ambígua

Uma gramática livre do contexto **G** é uma gramática ambígua se existe pelo menos uma palavra tal que possuam:

duas ou mais derivações mais à esquerda; ou

**definição 6.8** – Linguagem inherentemente ambígua  
Uma linguagem é dita uma *linguagem inherentemente ambígua* se qualquer gramática livre do contexto que a define é ambígua.

**definição 6.8** – Linguagem inherentemente ambígua  
Uma linguagem é dita uma linguagem inherentemente ambígua no contexto que a definição de ambiguidade.

LITERATURE 68

**Exemplo 8.8** – Em Guageon, frontemente ambíguo

卷之三

卷之九

**Exemplo 6.3** Linguagem inherentemente ambígua: contraexemplo  
Considere o exemplo 6.2 – Gramática livre do contexto: expressões aritméticas. A linguagem em questão não é inherentemente ambígua.

A construção de uma gramática não ambígua para esta linguagem é sugerida como exercício. O correto entendimento da solução deste exercício é especialmente importante, pois justifica a maneira aparentemente "estranha" de definir expressões na maioria das gramáticas de linguagens de programação. De fato, como já foi destacado, no desenvolvimento e na implementação de alguns tipos de algoritmos de reconhecimento, é conveniente que a gramática dada seja não ambígua.

64

possível simplificar alguns tipos de produções sem reduzir o poder de geração das gramáticas livres do contexto. Em geral, as simplificações de gramáticas são usadas na construção e otimização de algoritmos e na demonstração de teoremas.

5 seuiintes significantes ελασσοναριστες

- a** Derivação mais à esquerda:  
 $E \Rightarrow E+E \Rightarrow x+E \Rightarrow x+E*E \Rightarrow x+x*x \Rightarrow x+x*x$

**b** Derivação mais à direita:  
 $E \Rightarrow E+E \Rightarrow E+E*E \Rightarrow E+x*E \Rightarrow x+x*x \Rightarrow x+x*x$

**c** Símbolos inúteis. Exclusão de variáveis ou terminais não usados para gerar palavras;  
**d** Produções vazias. Exclusão de produções da forma  $A \rightarrow \epsilon$  (se a palavra vazia pertence à linguagem, é incluída uma produção vazia específica para tal fim);  
**e** Produções que substituem variáveis. Exclusão de produções da forma  $A \rightarrow B$ , ou seja, que simplesmente substituem uma variável por outra e, consequentemente, não adicionam qualquer informação de geracão de palavras

No texto que segue, são omitidas as provas de que os algoritmos de simplificação apresentados, de fato, atingem os objetivos propostos.

#### 6.4.1 símbolos inúteis

A exclusão de *símbolos inúteis* (símbolos não usados na geração de palavras de terminais) é realizada, excluindo-se as produções que fazem referência a esses símbolos, bem como o gramática. O algoritmo apresentado é dividido em duas etapas, como segue:

- a** *Etapa 1: qualquer variável gera terminais.* O algoritmo restringe o conjunto de variáveis analisando as produções da gramática a partir de terminais gerados. Inicialmente, considera-se todas as variáveis que geram terminais diretamente (exemplo:  $A \rightarrow a$ ). A seguir, são adicionadas, sucessivamente, as variáveis que geram terminais indiretamente (exemplo:  $B \rightarrow Ab$ ). acima, o algoritmo analisa as produções da gramática a partir do símbolo inicial. Inicialmente, considera exclusivamente o símbolo inicial. Após, sucessivamente, as produções da gramática são aplicadas, e os símbolos referenciados são adicionados aos novos conjuntos.
- b** *Etapa 2: qualquer símbolo é atingível a partir do símbolo inicial.* Após a execução da etapa acima, o algoritmo analisa as produções da gramática a partir do símbolo inicial. Inicialmente, considera exclusivamente o símbolo inicial. Após, sucessivamente, as produções da gramática são aplicadas, e os símbolos referenciados são adicionados aos novos conjuntos.

**definição 6.9 – Algoritmo: exclusão dos símbolos inúteis**

Seja  $G = (V, T, P, S)$  uma gramática livre do contexto. O algoritmo para exclusão dos símbolos inúteis é composto por duas etapas, como segue:

*Etapa 1: qualquer variável gera terminais. A gramática resultante desta etapa é:*

$$G_1 = (V_1, T, P_1, S)$$

na qual  $V_1 \subseteq V$  é construído conforme o algoritmo apresentado na figura 6.6. O conjunto  $P_1$  possui os mesmos elementos que  $P$ , excetuando-se as produções cujas variáveis não pertencem a  $V_1$ .

*Etapa 2: qualquer símbolo é atingível a partir do símbolo inicial.* A gramática resultante desta etapa é:

$$G_2 = (V_2, T_2, P_2, S)$$

na qual  $V_2 \subseteq V_1$  e  $T_2 \subseteq T$  são construídos conforme o algoritmo apresentado na figura 6.7 (suponha que  $\alpha, \beta \in (T \cup V_1)^*$ ):

O conjunto  $P_2$  possui os mesmos elementos que  $P_1$ , excetuando-se as produções cujos símbolos não pertencem a  $V_2$  ou  $T_2$ . □

No texto que segue, são omitidas as provas de que os algoritmos de simplificação apresentados, de fato, atingem os objetivos propostos.

$$\begin{aligned} T_2 &= \emptyset; \\ V_2 &= \{S\}; \\ \text{repita } &V_2 = V_2 \cup \{A \mid X \rightarrow \alpha A \beta \in P_1, X \in V_2\}; \\ &T_2 = T_2 \cup \{a \mid X \rightarrow \alpha a \beta \in P_1, X \in V_2\} \\ \text{até que os cardinais de } &V_2 \text{ e } T_2 \text{ não aumentem;} \end{aligned}$$

**figura 6.7** Algoritmo: exclusão dos símbolos inúteis – etapa 2.

Deve-se reparar que se as etapas acima forem executadas em ordem inversa (etapa 2 antes da etapa 1), o algoritmo pode não atingir o resultado esperado. Para demonstrar, é suficiente apresentar um contraexemplo, o que se sugere como exercício (lembre-se de que uma demonstração por contraexemplo é, de fato, uma demonstração por absurdo).

**exemplo 6.10 – Exclusão dos símbolos inúteis**

Considere a seguinte gramática livre do contexto:

$$\begin{aligned} G &= (\{S, A, B, C\}, \{a, b, c\}, P, S), \text{ na qual:} \\ P &= \{S \rightarrow aAa \mid bBb, A \rightarrow a \mid S, C \rightarrow c\} \end{aligned}$$

A exclusão dos símbolos inúteis é como segue:

*Etapa 1: qualquer variável gera terminais.* Considere a figura 6.8. A coluna “iteração” representa o número de ciclos do comando repita-até, e a coluna “variáveis”, o conjunto de variáveis construído após a iteração. Observe que o algoritmo para na terceira iteração, pois nenhuma variável foi adicionada ao conjunto.

A produção  $S \rightarrow bBb$  é excluída, pois  $B$  não pertence ao novo conjunto de variáveis; A gramática resultante desta etapa é a seguinte:

$$G_1 = (\{A, C, S\}, \{a\}, \{S \rightarrow aAa, A \rightarrow a \mid S, C \rightarrow c\}, S)$$

*Etapa 2: qualquer símbolo é atingível a partir do símbolo inicial.* Considere a figura 6.9. A produção  $C \rightarrow c$  é excluída, pois  $C$  e  $c$  não pertencem aos novos conjuntos de variáveis e terminais, respectivamente. A gramática resultante desta etapa é a seguinte:

$$G_2 = (\{S, A\}, \{a\}, \{S \rightarrow aAa, A \rightarrow a \mid S\}, S)$$

□

iteração	variáveis
início	$\emptyset$
1	$\{A, C\}$
2	$\{A, C, S\}$
3	$\{A, C, S\}$

**figura 6.8** Exclusão dos símbolos inúteis – etapa 1.

$$\begin{aligned} V_1 &= \emptyset; \\ \text{repita } &V_1 = V_1 \cup \{A \mid A \rightarrow \alpha \in P \text{ e } \alpha \in (T \cup V_1)^*\} \\ \text{até que o cardinal de } &V_1 \text{ não aumente;} \end{aligned}$$

**figura 6.6** Algoritmo: exclusão dos símbolos inúteis – etapa 1.

iteração	variáveis	terminais
início	$\{S\}$	$\emptyset$
1	$\{S, A\}$	{a}
2	$\{S, A\}$	{a}

figura 6.9 Exclusão dos símbolos inúteis – etapa 2.

#### 6.4.2 produções vazias

A exclusão de produções vazias (produções da forma  $A \rightarrow \epsilon$ ) pode determinar modificações diversas nas produções da gramática. O algoritmo é dividido em três etapas, como segue:

- a** *Etapa 1: variáveis que constituem produções vazias.* Considera, inicialmente, todas as variáveis que geram diretamente a palavra vazia (exemplo:  $A \rightarrow \epsilon$ ). A seguir, são determinadas, sucessivamente, as variáveis que indiretamente geram a palavra vazia (exemplo:  $B \rightarrow A$ ).
- b** *Etapa 2: exclusão de produções vazias.* Inicialmente, são consideradas todas as produções não vazias. A seguir, cada produção cujo lado direito possui uma variável que gera a palavra vazia, determina uma produção adicional, sem essa variável;
- c** *Etapa 3: geração da palavra vazia, se necessário.* Se a palavra vazia pertence à linguagem, então é incluída uma produção para gerar a palavra vazia.

**definição 6.10** – Algoritmo: exclusão das produções vazias

Seja  $G = (V, T, P, S)$  gramática livre do contexto. O algoritmo para exclusão das produções vazias é composto por três etapas, como segue:

- Etapa 1: variáveis que constituem produções vazias.* O algoritmo para construir o conjunto das variáveis que geram  $\epsilon$ , denotado por  $V_\epsilon$ , é apresentado na figura 6.10;
- Etapa 2: exclusão de produções vazias.* A gramática resultante desta etapa é:

$$G_1 = (V, T, P_1, S)$$

onde  $P_1$  é construído conforme o algoritmo apresentado na figura 6.11;

- Etapa 3: geração da palavra vazia, se necessário.* Se a palavra vazia pertence à linguagem, então a seguinte produção é incluída:

$$S \rightarrow \epsilon$$

resultando na seguinte gramática:

$$\begin{aligned} G_2 &= (V, T, P_2, S) \text{ onde:} \\ P_2 &= P_1 \cup \{S \rightarrow \epsilon\} \end{aligned}$$

$$\begin{aligned} V_\epsilon &= \{A \mid A \rightarrow \epsilon \in P\}, \\ \text{repita } V_\epsilon &= V_\epsilon \cup \{X \mid X \rightarrow X_1 \dots X_n \in P \text{ tal que } X_1, \dots, X_n \in V_\epsilon\} \\ \text{até } V_\epsilon &\text{ que o cardinal de } V_\epsilon \text{ não aumente;} \end{aligned}$$

figura 6.10 Algoritmo: exclusão das produções vazias – etapa 1.

$P_1 = \{A \rightarrow \alpha \mid A \rightarrow \alpha \in P \text{ e } \alpha \neq \epsilon\};$ repita para toda $A \rightarrow \alpha \in P_1$ , $X \in V_\epsilon$ tal que $\alpha = \alpha_1 X \alpha_2$ , $\alpha_1 \alpha_2 \neq \epsilon$ fazendo $P_1 = P_1 \cup \{A \rightarrow \alpha_1 \alpha_2\}$ até que o cardinal de $P_1$ não aumente;
--

figura 6.11 Algoritmo: exclusão das produções vazias – etapa 2.

**exemplo 6.11** – Exclusão das produções vazias

Considere a seguinte gramática livre do contexto:

$$\begin{aligned} G &= (\{S, X, Y\}, \{a, b\}, P, S), \text{ na qual:} \\ P &= \{S \rightarrow aXa \mid bXb \mid \epsilon, X \rightarrow a \mid b \mid Y, Y \rightarrow \epsilon\} \end{aligned}$$

A exclusão das produções vazias é como segue:

- Etapa 1: variáveis que constituem produções vazias.* O conjunto  $V_\epsilon$  é construído conforme ilustrado na figura 6.12;
- Etapa 2: exclusão de produções vazias.* O novo conjunto de produções é construído conforme ilustrado na figura 6.13. A gramática resultante desta etapa é a seguinte:

$$G_1 = (\{S, X, Y\}, \{a, b\}, \{S \rightarrow aXa \mid bXb \mid aa \mid bb, X \rightarrow a \mid b \mid Y\}, S)$$

- Etapa 3: geração da palavra vazia, se necessário.* Como a palavra vazia pertence à linguagem, a produção  $S \rightarrow \epsilon$  é incluída no conjunto de produções.

A gramática resultante é a seguinte:

$$G_2 = (\{S, X, Y\}, \{a, b\}, \{S \rightarrow aXa \mid bXb \mid aa \mid bb \mid \epsilon, X \rightarrow a \mid b \mid Y\}, S)$$

Observe que  $Y$ , originalmente um símbolo útil, resultou em um símbolo inútil. Ou seja, a exclusão de produções vazias gerou um símbolo inútil. De fato, não é qualquer combinação de simplificações de gramática que atinge o resultado desejado. Veja adiante a seção 6.4.4 – Simplificações combinadas.  $\square$

iteração	$V_\epsilon$
início	$\{S, Y\}$
1	$\{S, Y, X\}$
2	$\{S, Y, X\}$

figura 6.12 Exclusão das produções vazias – etapa 1.

iteração	produções
início	$\{S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid Y\}$
1	$\{S \rightarrow aXa \mid bXb \mid aa \mid bb, X \rightarrow a \mid b \mid Y\}$
2	$\{S \rightarrow aXa \mid bXb \mid aa \mid bb, X \rightarrow a \mid b \mid Y\}$

figura 6.13 Exclusão das produções vazias – etapa 2.

### 6.4.3 produções que substituem variáveis

Uma produção que substitui diretamente uma variável por outra, ou seja, do tipo  $A \rightarrow B$  não adiciona informação alguma em termos de geração de palavras, a não ser o fato de que, neste caso, a variável  $A$  pode ser substituída por  $B$ . Assim, se  $B \rightarrow \alpha$ , então a produção  $A \rightarrow B$  pode ser substituída por  $A \rightarrow \alpha$ . A generalização desta ideia é o algoritmo proposto, dividido em duas etapas, como segue:

- a** *Etapa 1: fecho transitivo de cada variável.* Entende-se por fecho transitivo de uma variável o conjunto de variáveis que podem substituí-la transitivamente. Por exemplo, se  $A \rightarrow B \in B \rightarrow C$ , então  $B$  e  $C$  pertencem ao fecho de  $A$ ;
- b** *Etapa 2: exclusão das produções que substituem variáveis.* Substitui as produções da forma  $A \rightarrow B$  por produções da forma  $A \rightarrow \alpha$ , na qual  $\alpha$  é atingível a partir de  $A$  através de seu fecho.

**definição 6.11 – Algoritmo: exclusão das produções que substituem variáveis**  
Seja  $G = (V, T, P, S)$  gramática livre do contexto. O algoritmo para exclusão das produções que substituem variáveis é composto por duas etapas, como segue:

- Etapa 1: fecho transitivo de cada variável.* O algoritmo para construir o fecho transitivo é apresentado na figura 6.14;
- Etapa 2: exclusão das produções que substituem variáveis.* A gramática resultante desta etapa é:

$$G_1 = (V, T, P_1, S)$$

na qual  $P_1$  é construído conforme o algoritmo apresentado na figura 6.15. □

**exemplo 6.12 – Exclusão das produções que substituem variáveis**

Considere a seguinte gramática livre do contexto:

$$\begin{aligned} G &= (\{S, X\}, \{a, b\}, P, S), \text{ onde:} \\ P &= \{S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid S \mid \varepsilon\} \end{aligned}$$

A exclusão da produção  $X \rightarrow S$  é como segue:

*Etapa 1: fecho transitivo da cada variável.*

$$\begin{aligned} FECHO-S &= \emptyset \\ FECHO-X &= \{S\} \end{aligned}$$

para toda  $A \in V$   
**FECHO-A** =  $\{B \mid A \neq B \text{ e } A \Rightarrow^+ B \text{ usando exclusivamente produções de } P \text{ da forma } X \rightarrow Y\}$ ;

**figura 6.14** Algoritmo: exclusão das produções que substituem variáveis – etapa 1.

```

 $P_1 = \{A \rightarrow \alpha \mid A \rightarrow \alpha \in P \text{ e } \alpha \notin V\};$ 
para toda  $A \in V \text{ e } B \in FECHO-A$ 
faça se  $B \rightarrow \alpha \in P \text{ e } \alpha \notin V$ 
então  $P_1 = P_1 \cup \{A \rightarrow \alpha\}$ ;
  
```

**figura 6.15** Algoritmo: exclusão das produções que substituem variáveis – etapa 2.

iteração	produções
início	$\{S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid \varepsilon\}$
S	$\{S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid \varepsilon\}$
X	$\{S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid \varepsilon \mid aXa \mid bXb\}$

**figura 6.16** Exclusão das produções que substituem variáveis – etapa 2.

**Etapa 2: exclusão das produções da forma  $A \rightarrow B$ .** Construção do conjunto de produções (a coluna “iteração” representa a execução do algoritmo para a variável referenciada) é conforme ilustrado na figura 6.16.

A gramática resultante é a seguinte:

$$G_1 = (\{S, X\}, \{a, b\}, \{S \rightarrow aXa \mid bXb, X \rightarrow a \mid b \mid \varepsilon \mid aXa \mid bXb\}, S)$$

### 6.4.4 simplificações combinadas

Deve-se observar que não é qualquer combinação de simplificações de gramática livre do contexto que atinge o resultado desejado. Por exemplo, em uma gramática sem símbolos inúteis, mas com produções que substituem variáveis, o algoritmo para excluir esse tipo de produção pode gerar símbolos inúteis (por quê?). Portanto, caso os algoritmos sejam combinados, a seguinte sequência de simplificação é recomendada:

- a** Exclusão das produções vazias;
- b** Exclusão das produções que substituem variáveis;
- c** Exclusão dos símbolos inúteis.

## 6.5 → formas normais

As formas normais estabelecem restrições rígidas na forma das produções, sem reduzir o poder de geração das gramáticas livres do contexto (excetuando-se a geração da palavra vazia). São usadas principalmente no desenvolvimento de algoritmos (com destaque para reconhecedores de linguagens) e na prova de teoremas.

As formas normais apresentadas são as seguintes (suponha que  $A$ ,  $B$  e  $C$  são variáveis, a é terminal e  $\varepsilon$  é uma palavra de variáveis):

- *Forma normal de Chomsky.* na qual as produções são da forma:

$$A \rightarrow BC \quad \text{ou} \quad A \rightarrow a$$