CAPÍTULO 5

LINGUAGENS LIVRES DE CONTEXTO

5.1. Introdução

Concluímos o capítulo anterior provando que há algumas linguagens que não são regulares. De facto há muitas linguagens não regulares, e são tantas que se podem classificar em várias famílias, como veremos posteriormente.

As linguagens livres de contexto constituem a família mais importante de linguagens, a ela pertencendo as linguagens de programação. As linguagens regulares são um caso particular de linguagens livres de contexto, constituindo uma sub-família destas.

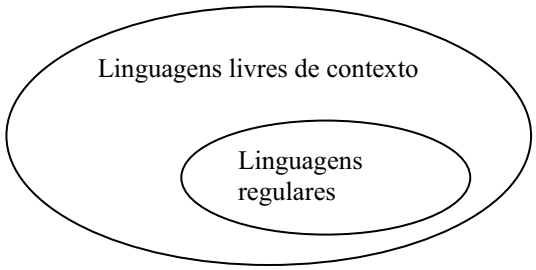


Figura 5.1.1. Relação entre as linguagens regulares e as linguagens livres de contexto.

Neste capítulo estudaremos as linguagens livres de contexto usando sobretudo as gramáticas (livres de contexto) e suas propriedades. Como veremos no Capítulo 6 os autómatos finitos não são capazes de reconhecer linguagens livres de contexto não-regulares por não terem memória.

5.2. Gramáticas livres de contexto

Nas gramáticas lineares que estudámos anteriormente existem duas restrições fundamentais:

- na parte esquerda das produções existe apenas uma variável
- na parte direita existe apenas uma variável na posição mais à esquerda ou mais à direita.

Relaxando a segunda restrição, e permitindo que aí existam diversas variáveis em qualquer posição, obtêm-se as gramáticas livres de contexto.

5.2.1. Definição e exemplos

Definição 5.2.1. Gramática livre de contexto e linguagem livre de contexto.

Uma gramática $G = (V, T, S, P)$ é chamada livre de contexto se todas as produções em P têm a forma

$$A \to x$$

em que $A \in V$ e $x \in (V \cup T)^*$, isto é, x é uma expressão qualquer composta por variáveis e/ou caracteres terminais ambos em número arbitrário.

Uma linguagem é livre de contexto se e só se existir uma gramática livre de contexto tal que $L = L(G)$.

Note-se que toda a gramática regular é também livre de contexto, e portanto toda a linguagem regular é também livre de contexto.

O nome de "livre de contexto" advém do seguinte facto: a substituição de uma variável na parte esquerda de uma produção pode fazer-se em qualquer altura em que essa variável apareça numa forma sentencial. Essa substituição não depende dos símbolos do resto da forma sentencial, ou seja, não depende do seu contexto. Esta característica resulta da existência de uma só variável na parte esquerda das produções. Se existissem duas como por exemplo em

$$AB \to x$$

a produção só se poderia aplicar quando aparecesse o par AB ou seja, quando fosse esse o contexto de A e de B. Esta gramática diz-se por isso dependente do contexto.

Exemplo 5.2.1.

A gramática $G=(\{S\}, \{a,b\}, S, P\}$ com produções

$$S \to aSa$$
$$S \to bSb$$
$$S \to \lambda$$

é livre de contexto.

Uma derivação possível,

$$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$$

Cada S introduz ou um par de a's ou um par de b's, sempre no meio da forma sentencial, e por isso no final a cadeia obtida é par e simétrica em relação ao seu centro.

A linguagem desta gramática é $L(G) = \{ww^R : w \in \{a,b\}^*\}$, uma linguagem livre de contexto não regular. Tente o leitor desenhar para ela um autómato finito.

Exemplo 5.2.2.

A gramática G com produções

$$S \rightarrow abB$$
$$A \rightarrow aaBb$$
$$B \rightarrow bbAa$$
$$A \rightarrow \lambda$$

é livre de contexto.

Uma derivação dela:

$S \Rightarrow abB \Rightarrow abbbAa \Rightarrow abbbaaBba \Rightarrow abbbaabbAaba \Rightarrow abbbaabbaaBbaba \Rightarrow$
$abbbaabbaabbAababa \Rightarrow abbbaabbaabbaaBbababa \Rightarrow abbbaabbaabbaabbAababababa$
$\Rightarrow abbbaabbaabbaabbaaBbabababa \Rightarrow abbbaabbaabbaabbaabbAabababababa \Rightarrow$
$abbbaabbaabbaabbaabbabababababa$

Não é fácil, olhando para as produções, generalizar a forma das cadeias obtidas pela gramática. No entanto, fazendo uma análise experimental, produzindo cadeias de vários comprimentos, verifica-se que $L(G) = \{ ab(bbaa)^n bba(ba)^n : n \geq 0 \}$.

Exemplo 5.2.3.

Mostrar que a linguagem $L = \{a^n b^m : n \neq m \}$ é livre de contexto. Note-se que pela definição da linguagem o número de a's tem que ser diferente do número de b's.

Para isso tem que se encontrar uma gramática livre de contexto que a produza. Para o caso $n = m$ viu-se no exemplo 1.3.2 do Cap. 1, obtendo-se

$$G = (\{S\}, \{a,b\}, S, P)$$

$$S \rightarrow aSb|\lambda$$

Seja agora o caso $n > m$.

Primeiro forma-se uma cadeia com um número igual de *a*'s e de *b*'s; de seguida acrescente-se um *a* adicional à esquerda. Para isso usam-se aquelas produções do caso $n=m$ e além disso uma para introduzir um ou mais *a*'s adicionais:

$$S \rightarrow AS_1,$$

$$S_1 \rightarrow aS_1b \mid \lambda,$$

$$A \rightarrow aA \mid a$$

Note-se que a segunda produção gera um número igual de *a*'s e de *b*'s. A última acrescenta *a*'s à esquerda. A primeira, ao criara o A logo de início, torna possível a última.

Esta gramática é não linear porque na primeira produção aparecem duas variáveis no lado direito. Mas é livre de contexto porque do lado esquerdo aparece só uma variável.

Tomemos agora o caso $n < m$. Aqui temos uma situação contrária à anterior: introduzem-se tantos *a*'s como *b*'s e depois introduzem-se à direita um ou mias *b*'s. Teremos as produções

$$S \rightarrow S_1B,$$
$$S_1 \rightarrow aS_1b \mid \lambda,$$
$$B \rightarrow Bb \mid b.$$

Para o caso $n \neq m$, juntam-se os dois conjuntos de produções, obtendo-se:

$$S \rightarrow AS_1 \mid S_1B,$$
$$S_1 \rightarrow aS_1b \mid \lambda,$$
$$A \rightarrow aA \mid a,$$
$$B \rightarrow Bb \mid b.$$

Note-se que para juntar os dois conjuntos de produções basta pôr as duas primeiras alternativas a partir de *S*.

Exemplo 5.2.4

Considere-se a gramática com produções

$$S \rightarrow aSb \mid SS \mid \lambda$$

Trata-se de uma gramática livre de contexto e não linear (porquê?).

Algumas cadeias geradas:

$$S \Rightarrow SS \Rightarrow SSSS \Rightarrow aSbaSbaSbaSb \Rightarrow abababab$$

$$S \Rightarrow aSb \Rightarrow aSSb \Rightarrow aaSbSb \Rightarrow aaaSbbaSbb \Rightarrow aaaSSbbaaSbbb$$
$$\Rightarrow aaaaSbaSbbbaaSbbb \Rightarrow aaaababbbaabb.$$

Gera cadeias com um número de *a*'s igual ao número de *b*'s (porquê ?) e em que o número de *a*'s em qualquer prefixo de qualquer cadeia é maior ou igual ao nº de *b*'s, i.e.

$$L = \{w \in \{a,b\}^* : n_a(w) = n_b(w) \text{ e } n_a(v) \geq n_b(v), v \text{ um prefixo qualquer de } w\}$$

e em que o número de *b*'s em qualquer sufixo de qualquer cadeia é maior ou igual ao nº de *a*'s, i.e.

$$L = \{w \in \{a,b\}^* : n_a(w) = n_b(w) \text{ e } n_a(u) \leq n_b(u), u \text{ um sufixo qualquer de } w\}$$

Substituindo *a* por parêntese à esquerda e *b* por parêntese à direita, obtém-se a linguagem para descrever a regra dos parênteses nas linguagens de programação. Ela contém por exemplo os casos (), (()), ((()())), ()()()(); etc. Esta situação é dada pela gramática

$$G=(\{S\}, \{(,)\}, S, P)$$

com produções

$$S \rightarrow (S) \mid SS \mid \lambda$$

Uma derivação será

$$S \Rightarrow (S) \Rightarrow (SS) \Rightarrow ((S)S) \Rightarrow (( )S) \Rightarrow (( ))$$

Neste exemplo os parênteses fazem parte do alfabeto da linguagem. Já no capítulo 1 vimos que um alfabeto pode ser composto por qualquer tipo de objectos.

Exemplo 5.2.5

Palíndromos em {0, 1}, (Hopcroft, 170).

A linguagem dos palíndromos pode-se definir por indução:

Base da indução: λ, 0 e 1 são palíndromos (palíndromos elementares)

Indução: se *w* é um palíndromo, também o são 0*w*0 e 1*w*1.

Nenhuma cadeia é um palíndromo de 0's e 1's a menos que seja formada a partir destas regras, que se podem traduzir pelas produções seguintes:

1. $P \rightarrow \lambda$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

Exemplo 5.2.6

Seja o alfabeto $\Sigma=\{x, y, z, +, -, (,)\}$. Encontrar uma gramática que gere todas as expressões aritméticas possíveis com este alfabeto, com por exemplo { *x, y, z, x+y, x-y, x+(y-z), ...*}.

Uma solução possível:

$G=(\{E,V\}, \{ x, y, z, +, -, (,), E, P)$

Em que o conjunto P é composto pelas seguintes 7 produções

1. $E \rightarrow E + E$
2. $E \rightarrow E - E$
3. $E \rightarrow (E)$
4. $E \rightarrow V$
5. $V \rightarrow x$

6. $V \rightarrow y$

7. $V \rightarrow z$

Como se poderá gerar x+*(y-z)* ?

$E \Rightarrow E + E$ (prod. 1)

$\Rightarrow V + E$ (prod. 4)

$\Rightarrow x + E$ (prod. 5) $\Rightarrow x + (E)$ (prod. 3)

$\Rightarrow x + (E - E)$ (prod. 2)

$\Rightarrow x + (V - E)$ (prod. 4)

$\Rightarrow x + (y - E)$ (prod. 6)

$\Rightarrow x + (y - V)$ (prod. 4)

$\Rightarrow x + (y - z)$ (prod. 7)

Exercício 5.2.1.

Considere a gramática com as produções

$S \rightarrow 0B \mid 1A$

$A \rightarrow 0 \mid 0S \mid 1AA$

$B \rightarrow 1 \mid 1S \mid 0BB$

Gere cinco cadeias com a gramática. Que linguagem lhe está associada?

5.2.2. Derivação pela extrema direita e pela extrema esquerda

Nas gramáticas livres de contexto não lineares aparece mais do que uma variável na parte direita das produções. Tem-se por isso uma escolha na ordem pela qual se substituem as variáveis. Por exemplo,

$G=(\{A,B,S\}, \{a,b\}, S, P\}$ com produções

1. $S \rightarrow AB$
2. $A \rightarrow aaA$
3. $A \rightarrow \lambda$
4. $B \rightarrow Bb$

5. $B \to \lambda$

Exemplo de derivação (o número da produção aplicada está indicado sobre a seta):

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aaBb \overset{5}{\Rightarrow} aab$$

$$S \overset{1}{\Rightarrow} AB \overset{4}{\Rightarrow} ABb \overset{2}{\Rightarrow} aaABb \overset{5}{\Rightarrow} aaAb \overset{3}{\Rightarrow} aab$$

A linguagem correspondente é $L(G) = \{a^{2n}b^m: n \geq 0, m \geq 0\}$

Nestas duas derivações obtém-se o mesmo resultado, usando precisamente as mesmas derivações mas por ordem diferente. Será sempre assim? Se se usam as mesmas derivações, ainda que por diferente ordem, introduzem-se os mesmos caracteres nas formas sentenciais e na sentença final, mas pode acontecer de diferentes ordens de aplicação das produções resultem em cadeias diferentes. Por exemplo as produções

1. $S \to aSb$
2. $S \to bSa$
3. $S \to \lambda$

A produção 1-2-3 dá

$$S \overset{1}{\Rightarrow} aSb \overset{2}{\Rightarrow} abSab \overset{3}{\Rightarrow} abab$$

e a produção 2-1-3

$$S \overset{2}{\Rightarrow} bSa \overset{1}{\Rightarrow} baSba \overset{3}{\Rightarrow} baba$$

dá uma cadeia diferente. Portanto a ordem das produções importa. Neste caso aplicando a produção 1 em primeiro lugar produzem-se cadeias que começam por *a*, e aplicando a 2 em primeiro lugar cadeias que começam por *b*.

Definição 5.2.2

Uma derivação diz-se pela **extrema esquerda** se em cada passo se substitui a variável mais à esquerda na forma sentencial.

Se se substituiu a variável mais à direita, a derivação diz-se pela **extrema direita**.

Exemplo 5.2.7

Seja a gramática com as produções

$$S \to aAB,$$
$$A \to bBb,$$
$$B \to A \mid \lambda$$

Então

$$S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb$$

é uma derivação pela direita e

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbbB \Rightarrow abbbb$$

é uma derivação pela esquerda.

5.2.3.Árvores de derivação (*parse trees*)

As árvores de derivação são uma alternativa à escrita das produções, permitindo uma visualização gráfica do processo de geração de cadeias. Por outro lado, lidas de forma inversa, permitem reconstruir as produções a partir da cadeia.

Um árvore de derivação é uma árvore ordenada em que

- os vértices iniciais e intermédios são as variáveis da gramática, etiquetados pelo lado esquerdo das produções
- os filhos de um vértice representam os lados direito correspondente ao nó pai.

Por exemplo a árvore da Fig. 5.2.1corresponde às produções:

$$S \to abABc$$
$$A \to \lambda$$
$$B \to c$$

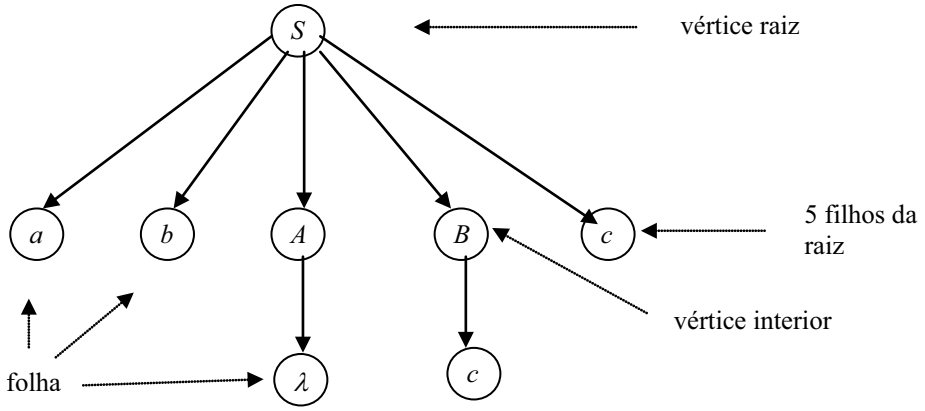


Figura 5.2.1. Uma árvore de derivação.

A partir da raiz aplica-se a primeira produção, dando 5 filhos, dos quais 3 são folhas e 2 são vértices interiores (variáveis). Depois do vértice *A* resulta a folha *c* pela segunda produção e do vértice *B* a folha λ pela terceira produção. Na árvore nada indica a ordem de aplicação das segunda e terceira produções.

Uma árvore de derivação inicia-se na raiz, com o símbolo inicial (em geral *S*), e termina em folhas que são terminais (símbolos do alfabeto). Em cada nível mostra como se substitui cada variável nas derivações. Quando um vértice contém um símbolo terminal, dele não parte qualquer aresta. Apenas dos vértices com variáveis, chamados vértices interiores (não raiz), partem arestas.

Vejamos a definição formal.

Definição 5.2.3. Árvore de derivação

Seja $G = (V, T, S, P)$ uma gramática livre de contexto. Uma árvore ordenada é uma **árvore de derivação** para a gramática *G* se e só se tiver as seguintes propriedades:

1. A raiz é etiquetada pelo símbolo inicial, em geral *S.*

2. Todas as folhas têm uma etiqueta de $T \cup \{\lambda\}$, isto é, um símbolo terminal

3. Todos os vértices interiores (vértices que não são folhas) têm uma etiqueta de *V*, i.e., uma variável.

4. Se um vértice tem uma etiqueta $A \in V$, e se os seus filhos são etiquetados (da esquerda para a direita) $a_1, a_2, \ldots, a_n$ então *P* deve conter a produção da forma

$$A \rightarrow a_1\, a_2 \ldots, a_n$$

em que a_i pode ser um símbolo terminal ou uma variável.

5. Uma folha etiquetada λ não tem irmãs, i.e., um vértice com um filho λ não pode ter outros filhos.

Se uma árvore verifica as propriedades 3, 4 e 5, mas a propriedade 1 não se verifica necessariamente e a propriedade 2 é substituída por

2a. Todas as folhas têm etiquetas de $V \cup T \cup \{\lambda\}$, i.e., uma variável ou um símbolo terminal,

diz-se uma **árvore de derivação parcial**, sendo parte de uma árvore maior. Isto é, se extrairmos de uma árvore de derivação uma sub-árvore interior, esta será uma árvore de derivação parcial, podendo ter variáveis nas folhas. Uma árvore de derivação parcial deriva formas sentenciais e pode não derivar cadeias terminais. Pelo contrário uma árvore de derivação total dá sempre cadeias terminais. Uma árvore de derivação é por defeito total, e portanto não é necessário adjectivá-la.

Lendo as folhas da árvore, da esquerda para a direita, omitindo quaisquer λ que se encontrem, obtém-se o **fruto (*yield*)** da árvore. O fruto é a cadeia de terminais obtida quando se percorre a árvore de cima para baixo, tomando sempre o ramo inexplorado mais à esquerda.

Exemplo 5.2.8

Seja a gramática $G=(\{S,A,B\}, \{a,b\}, S, P\}$ com produções

$$S \rightarrow aAB,$$
$$A \rightarrow bBb,$$
$$B \rightarrow A \mid \lambda$$

A árvore de derivação parcial da Fig. 5.2.2 corresponde a

$$S \Rightarrow aAB \Rightarrow abBbB$$

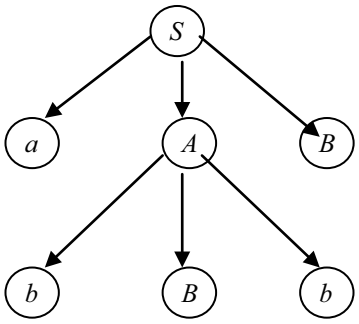


Figura 5.2.2. Àrvore de derivação parcial

A árvore de derivação Fig. 5.2.3 corresponde à derivação pela esquerda

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbA \Rightarrow abbbBb \Rightarrow abbbb$$

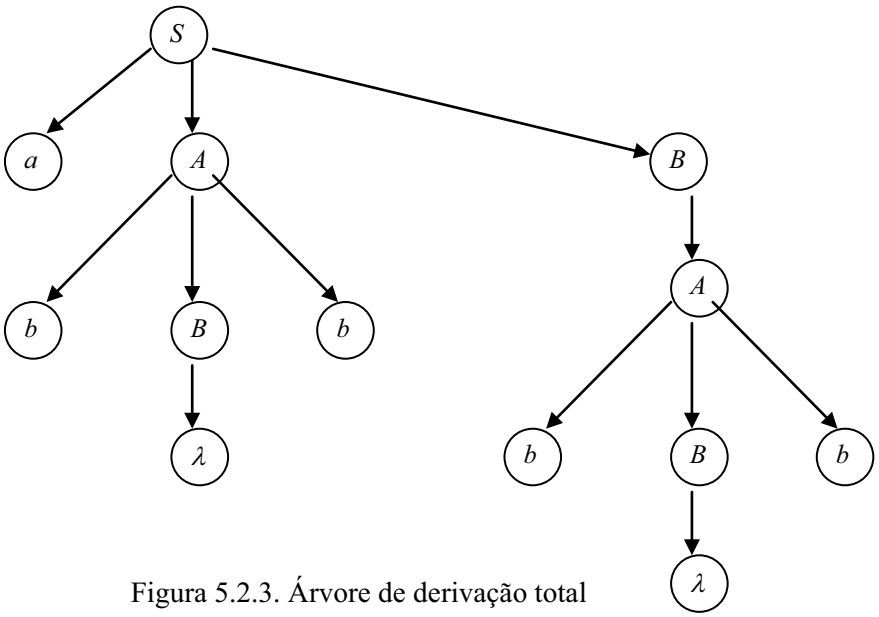


Figura 5.2.3. Árvore de derivação total

ou à derivação pela direita

$$S \Rightarrow aAB \Rightarrow aAA \Rightarrow aAbBb \Rightarrow aAbb \Rightarrow abBbbb \Rightarrow abbbb$$

Uma outra árvore de derivação Fig 5.2.4 corresponde a (pela esquerda)

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbB \Rightarrow abb$$

ou a (pela direita)

$$S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abb$$

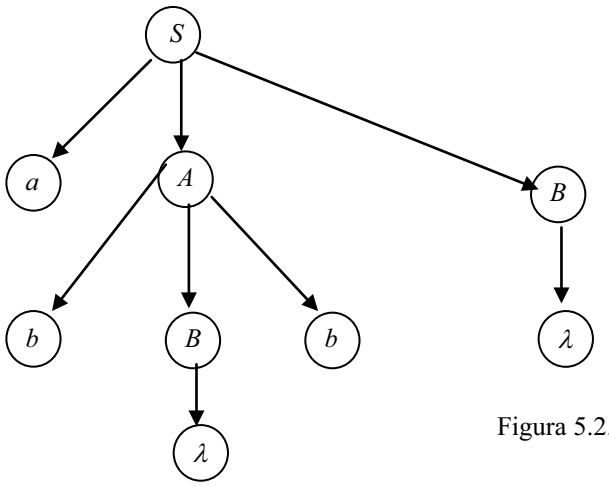


Figura 5.2.4. Outra árvore de derivação

Podemos ver que uma árvore de derivação está ligada a uma derivação particular e não é um esquema geral de derivações para uma dada gramática. Cada cadeia terá a sua árvore de derivação própria. Além disso a árvore exprime uma derivação pela direita ou uma derivação pela esquerda.

Dada uma gramática livre de contexto $G=(V, T, S, P)$, para toda a cadeia $w \in L(G)$, existe uma árvore de derivação de G cujo fruto é w. Basta desenhá-la aplicando as produções que geraram w.

Inversamente, o fruto de qualquer árvore de derivação, desenhada de acordo com as produções de P, pertence $L(G)$.

Por outro lado se t_G é alguma árvore de derivação parcial de G cuja raiz está etiquetada por S, então o fruto de t_G é uma forma sentencial da gramática G.

O leitor pode ver uma prova formal por indução destas afirmações em Linz p. 132.

As árvores de derivação evidenciam as produções usadas na obtenção de uma qualquer cadeia, mas não explicitam a ordem da sua aplicação. Qualquer cadeia $w \in L(G)$ tem uma derivação pela extrema esquerda e uma derivação pela extrema direita. Este facto interessante constata-se observando que a árvore tanto pode ser desenhada da esquerda para a direita (produções pela esquerda) como da direita para a esquerda (produções pela direita).

As árvores de derivação, na literatura de língua inglesa (por exemplo em Hopcroft, ou em Sipser), são chamadas *parse trees*. Elas mostram como os símbolos terminais de uma cadeia se agrupam em sub-cadeias, cada uma das quais pertence à linguagem de uma das variáveis da gramática, isto é, ao conjunto das cadeias geráveis a partir de uma variável da gramática

num vértice interior. Nos compiladores as árvores de *parsing* são a estrutura de dados escolhida para representar o código fonte de um programa. É essa estrutura de dados assim definida que facilita a tradução do código fonte em código executável, ao permitir que sejam usadas funções recursivas para o processo de tradução. Na disciplina de compiladores terá o leitor oportunidade de aprofundar esta questão. A palavra derivação pode, até certo ponto, ser usada como tradução de *parsing*. No entanto *parsing* é não a derivação em si mesma mas a sua procura e por isso mantém-se o termo *parsing* original, à falta de melhor tradução.

5.3. Parsing

Dada uma gramática qualquer, sabemos gerar cadeias da sua linguagem, aplicando as suas produções. As cadeias geradas pertencem à linguagem da gramática.

Mas podemos agora considerar a questão inversa. Dada uma cadeia w, como saber se ela pertence a uma certa linguagem $L(G)$? Este é o problema da **pertença**, que já conhecemos das linguagens regulares.

E se a cadeia pertence à linguagem, qual a sequência de produções que a gerou? Este é o problema de **parsing** : encontrar uma sequência de produções pelas quais se deriva $w \in L(G)$. O conceito de parsing, uma palavra difícil de traduzir, aplica-se no estudo das linguagens naturais (ver por exemplo em http://nltk.sourceforge.net/doc/en/parse.html) e nas linguagens de computador. Os compiladores quando analisam um programa escrito numa dada linguagem, fazem precisamente isso: procuram a sequência de produções da gramática da linguagem que levou ao programa concreto. Se não a encontram, indicam erro. Se a encontram, a compilação é bem sucedida.

5.3.1. Parsing e ambiguidade

O parsing é uma operação delicada, que tem absorvido uma parte significativa da investigação em computação a nível mundial. Se imaginarmos que os programas de computador são cada vez maiores, fazer o seu parsing em tempo útil (para o compilar depois) é um desafio para o qual ainda hoje se procuram respostas melhoradas. As técnicas de parsing baseiam-se na teoria de grafos e seus algoritmos (por isso a teoria de grafos é um dos mais importantes temas matemáticos das ciências da computação).

Dada uma cadeia w em $L(G)$ pode-se fazer o seu **parsing por procura exaustiva**, construindo sistematicamente todas as derivações possíveis (pela esquerda, por exemplo) e verificando se se obtém a procurada w ou não. Esta técnica executa-se por voltas sucessivas, até que se encontre uma resposta. Procede-se do modo seguinte.

1ª volta - analisar as produções do tipo

$$S \to x$$

encontrando todas as formas sentenciais e cadeias x que se podem obter de S em um passo.

Alguma é *w?* Se sim parar

2ª volta – aplicar todas as produções possíveis à variável mais à esquerda em cada forma sentencial x da 1ª volta, obtendo-se um conjunto de formas sentenciais alcançáveis em dois passos.

Alguma é w ?

3ª volta – aplicar todas as produções possíveis à variável mais à esquerda em cada forma sentencial obtida da 2ª volta, obtendo um novo conjunto de formas sentenciais alcançáveis em três passos.

Alguma é w ?

e assim sucessivamente.

Se $w \in L(G)$, deve ter uma derivação de extrema esquerda de comprimento finito. Por isso este método exaustivo há-de encontrar a solução, se ela existir.

Trata-se de um método de parsing de cima-para-baixo, que pode ser visto simplesmente como a construção da árvore de derivação para baixo a partir da raiz.

Exemplo 5.3.1
Seja a gramática com as produções

$$S \to SS \mid aSb \mid bSa \mid \lambda.$$

Fazer o parsing da cadeia $w = aabb$.

Vejamos como se aplica a procura exaustiva, desenhando a árvore da procura.

1ª volta: inicializar a produção da gramática a partir da raiz *S*

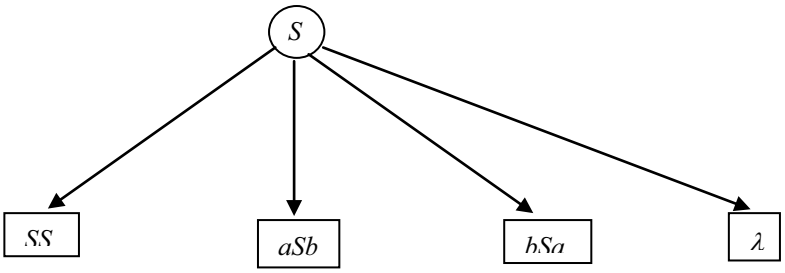


Figura 5.3.1. Primeira volta do parsing de *aabb.*

Nenhuma cadeia produzida é *w*. Podem-se desde já eliminar as folhas de *bSa* (dado que *w* se inicia por *a*) e λ (dado que |*w*|>0). Em cada etapa segue-se apenas por onde possa estar a solução. Caminhos que se sabe serem estéreis não de prosseguem.

Note-se a diferença entre árvore de derivação (ou árvore de *parsing*) , que vimos atrás, e esta árvore de procura de cadeias. Os filhos da raiz são as produções alternativas a partir de *S,* e portanto as suas etiquetas são formas sentenciais ou sentenças (daí a sua forma rectangular para distinguir dos círculos da árvore de derivação). Por outro lado esta árvore de procura é bastante genérica e não é afecta apenas a uma cadeia (embora se vá particularizando à media que se prossegue, pelo abandono de caminhos inexequíveis).

2ª volta: a partir dos vértices restantes da etapa 1, gerar todos os filhos possíveis, usando uma e uma só produção pela esquerda. Por exemplo em *SS* substitui-se apenas o primeiro *S*.

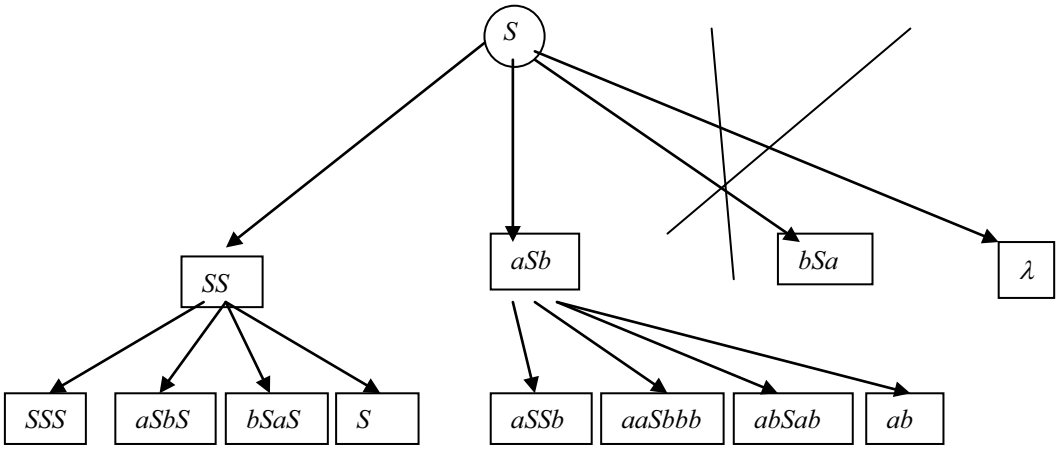


Figura 5.3.2. Segunda volta do parsing exaustivo

Nenhuma cadeia é *w*. Podem-se desde já eliminar as folhas que começam por *b* ou cujo segunda carácter é *b*.

3ª volta: prossegue-se a partir de cada vértice da 2ªvolta, com produções pela esquerda.

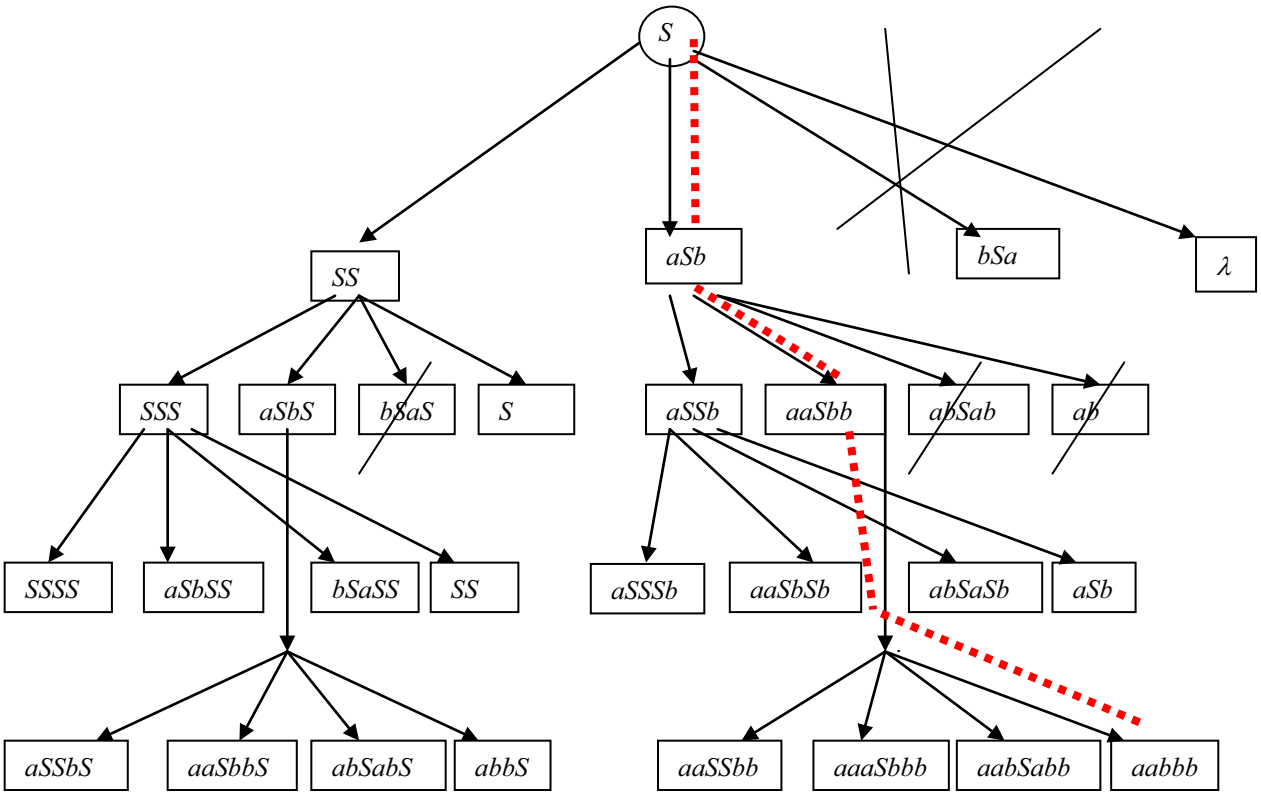


Figura 5.3.3. Terceira e última volta do parsing de *aabb*

E encontrou-se finalmente a cadeia *aabb*. A sua derivação está identificada. O seu parsing está concluído. Ela foi produzida pelo caminho assinalado a tracejado.

$$S \Rightarrow aSb \Rightarrow aaSbbb \Rightarrow aabb$$

Como se pode ver, o parsing exaustivo tem algumas desvantagens. Se a cadeia fosse muito grande, a árvore de procura seria também muito grande. Além disso é um processo fastidioso, parecido com os métodos de "força bruta" que consistem em experimentar todas as alternativas possíveis para um problema. Por isso é um processo pouco eficiente, e nenhum compilador o usa hoje em dia. Tem apenas um interesse didáctico, ajudando-nos a perceber a essência do processo de parsing.

Tem além disso um problema maior: pode nunca terminar se a cadeia não está na linguagem (verificar neste caso o que acontece para *w= abb*), a menos que se introduza um mecanismo de paragem ao fim de certo tempo.

Como evitar a não paragem do parsing?

Se admitirmos que uma cadeia, por muito grande que seja, é finita em tamanho, o parsing exaustivo acabará por parar, desde que de etapa para etapa aumente o tamanho das formas sentenciais. Se na gramática existirem produções do tipo

$$S \to \lambda \quad \text{(reduz o tamanho da forma sentencial)}$$

$$A \to B \text{ (mantém o tamanho da forma sentencial}$$

esse facto não está assegurado. Portanto produções deste tipo devem ser eliminadas, introduzindo-se assim algumas restrições na forma da gramática. Tal não acarreta problemas adicionais dado que tais restrições não diminuem significativamente o poder das gramáticas livres de contexto. No Capítulo 6 estudaremos técnicas de eliminar produções daqueles tipos (chamadas nulas e unitárias) nas gramáticas livres de contexto.

Exemplo 5.3.2.

A gramática com produções

$$S \to SS \mid aSb \mid bSa \mid ab \mid ba$$

obedece à restrição acima mencionada. Pode-se verificar que gera a mesma linguagem do exemplo anterior, mas agora sem a cadeia vazia.

Uma produção possível (de extrema esquerda):

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abSabS \Rightarrow abababS \Rightarrow abababba$$

Outra produção (de extrema direita) :

$$S \Rightarrow SS \Rightarrow SaSb \Rightarrow SabSab \Rightarrow SabbSaab \Rightarrow Sabbabaab \Rightarrow baabbabaab$$

Repare-se que as sucessivas formas sentenciais aumentam de tamanho pelo menos em uma unidade, e portanto ao fim de n voltas elas terão pelo menos $n+1$ caracteres.

Considere-se o seguinte exemplo para nos apoiar na análise.

Uma outra produção possível pela esquerda:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abSabS \Rightarrow abSSabS \Rightarrow abbSaSabS \Rightarrow abbaSbaSabS$$
$$\Rightarrow abbaabbaSabS \Rightarrow abbaabbabaabS \Rightarrow abbaabbabaabba$$

As formas sentenciais vão aumentando em comprimento e contêm variáveis e símbolos terminais.

Outra produção

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow SSSS \Rightarrow SSSSS \Rightarrow SSSSSS \Rightarrow SSSSSSS \Rightarrow abSSSSSS \Rightarrow abbaSSSSS$$
$$\Rightarrow abbabaSSSS \Rightarrow abbabaabSSS \Rightarrow abbabaabbaSS \Rightarrow abbabaabbabaS$$
$$\Rightarrow abbabaabbabaab$$

Também aqui as formas sentenciais vão aumentando de comprimento. Inicialmente contêm apenas variáveis, mas a partir da 6ª produção elas começam a ser substituídas por caracteres do alfabeto (símbolos terminais) até que a cadeia seja obtida.

Dada uma cadeia $w \in \{a,b\}^+$, o método de parsing exaustivo termina sempre em não mais do que $2|w|$ voltas. Depois de (no máximo, caso extremo) $|w|$ voltas, teremos enumerado todas as formas sentencias com $|w|$ caracteres. Essas formas sentenciais têm variáveis e símbolos terminais. No caso limite podem ter apenas $|w|$ variáveis e nenhum símbolo terminal. A partir daqui é necessário substituir cada uma das variáveis para se obter uma cadeia de símbolos terminais. Na pior das hipóteses, substitui-se uma variável por um símbolo terminal de cada vez, o que implica mais $|w|$ voltas. Assim, na pior das hipóteses, o parsing leva $2|w|$ voltas a completar-se (ou o parsing é encontrado ou a cadeia não pertence à linguagem). Daí o teorema seguinte.

Teorema 5.3.1. Considere-se uma gramática livre de contexto que não tem qualquer produção da forma

$$A \rightarrow \lambda$$
$$A \rightarrow B$$

em que $A, B \in V$. Então o parsing exaustivo pode ser feito por um algoritmo que, para qualquer $w \in \Sigma^*$, ou produz o parsing de w, ou conclui que não é possível qualquer parsing para w.

Qual o número máximo de formas sentenciais que se podem obter pelo parsing exaustivo ?

A gramática tem um número de produções distintas igual a $|P|$.

Na 1ª volta obtêm-se $|P|$ formas sentenciais.

Na 2ª volta obtêm-se $|P|\times|P|=|P|^2$ formas sentenciais, no máximo.

Na 3ª volta obtêm-se $|P|^2\times|P|=|P|^3$ formas sentenciais, no máximo.

...

Na $2|w|$ª volta obtêm-se $|P|^{2|w|}$ formas sentenciais, no máximo.

Somando agora ao longo de todas as voltas, obtém-se o limite superior para o número de formas sentenciais que se podem obter:

$$F = |P| + |P|^2 + |P|^3 + ... + |P|^{2|w|}$$

A última parcela evidencia que o comprimento da cadeia é expoente do último termo da soma. Prova-se assim que o trabalho de busca cresce exponencialmente com o comprimento da cadeia, podendo tornar proibitivo o custo (computacional) do método.

Este é um problema central em teoria da computação: encontrar métodos de parsing que tenham menor complexidade computacional (isto é, que sejam mais rápidos). Ainda hoje é um importante tema de investigação. Repare-se que para um compilador a cadeia em causa é o programa completo. Fazer o parsing da cadeia é verificar se o programa obedece às produções da gramática da linguagem, isto é, se está bem escrito.

Pode-se demonstrar que para quaisquer gramáticas livres de contexto existe um algoritmo que faz o parsing de qualquer cadeia w num tempo proporcional a $|w|^3$. É melhor do que crescimento exponencial, mas ainda muito ineficiente (um compilador que fosse por aí, demoraria um tempo exagerado a compilar um programa). O que ainda se busca actualmente é um algoritmo de parsing que demore um tempo linear com o comprimento da cadeia para qualquer gramática livre de contexto.

Felizmente para algumas gramáticas especiais já se encontrou um método linear. São as chamadas gramáticas simples.

Definição 5.3.2. Uma gramática livre de contexto $G = (V, T, S, P)$ é uma **gramática simples** ou uma **s-gramática** se todas as suas produções se iniciam por um símbolo terminal seguido de zero ou mais variáveis, ou seja, se são da forma

$$A \rightarrow ax$$

em que $A \in V$, $a \in T$, $x \in V^*$, e qualquer par (A, a) aparece no máximo uma vez em P.

Em cada produção substitui-se uma variável por um símbolo terminal e uma combinação de variáveis. Assim aumenta-se, por cada produção, exactamente em uma unidade o número de caracteres terminais em cada forma sentencial. Ao fim de $|w|$ produções já se introduziram $|w|$ símbolos terminais e portanto já se encontrou a cadeia se ela pertence à linguagem. Pode-se parar ao fim de $|w|$ voltas e por isso o número de voltas cresce linearmente com $|w|$.

Exemplo 5.3.3.

A gramática com as produções

$$S \rightarrow aS \mid bSS \mid c$$

é uma s-gramática.

Uma produção possível (à direita):

$$S \Rightarrow aS \Rightarrow abSS \Rightarrow abSaS \Rightarrow abSac \Rightarrow abcac$$

Para se introduzir a, b ou c na forma sentencial há apenas uma possibilidade em cada volta.

A gramática com produções

$$S \rightarrow aS \mid bSS \mid aSS \mid c$$

não é uma s-gramática porque o par (S, a) aparece nas duas produções $S \rightarrow aS$ e $S \rightarrow aSS$.

Uma produção possível:

$$S \Rightarrow aS \Rightarrow abSS \Rightarrow abSaSS \Rightarrow abSaSbSS \Rightarrow abSaSbSc \Rightarrow abSaSbcc \Rightarrow abSacbcc \Rightarrow abcacbcc$$

Aqui também o número de caracteres terminais aumenta em uma unidade por cada produção. Para se introduzir *a* na forma sentencial há duas possibilidades (duas produções possíveis) e daí o facto de o processo de parsing ser mais longo.

Muitas características de linguagens de programação podem ser descritas por s-gramáticas. Um parsing numa s-gramática pode ser feito em $|w|$ voltas, e por isso o seu tempo é linear com o tamanho da cadeia.

Exemplo 5.3.4

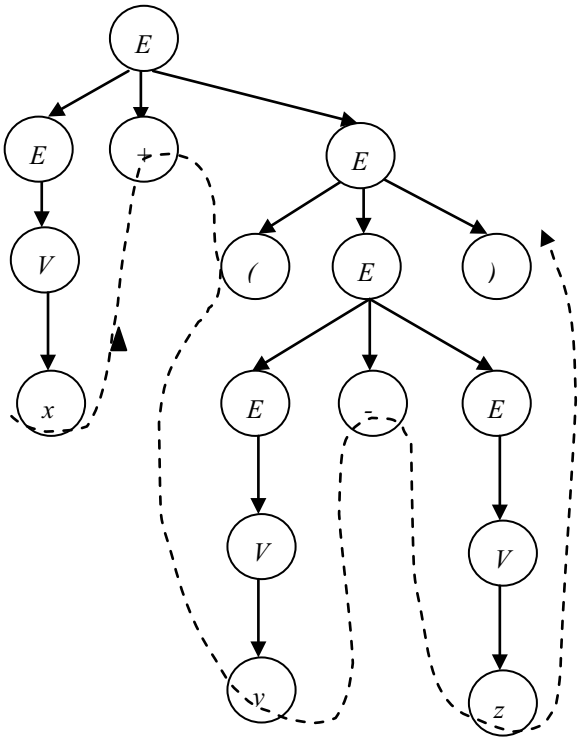
Seja a gramática livre de contexto já encontrada no exemplo 5.2.6 (não é uma s-gramática).

$$E \rightarrow E + E$$
$$E \rightarrow E - E$$
$$E \rightarrow (E)$$
$$E \rightarrow V$$
$$V \rightarrow x$$
$$V \rightarrow z$$
$$V \rightarrow y$$

Como se pode derivar a cadeia *x+(y-z)*?

Desenhe-se a árvore de derivação, na Figura 5.3.4.

A cadeia *x + (y - z)* encontra-se lendo as folhas da árvore da esquerda para a directa. Ela é o fruto da árvore.



A derivação da cadeia *x+(y-z) será*

$E \Rightarrow E + E$
$\Rightarrow V + E$
$\Rightarrow x + E$
$\Rightarrow x + (E)$
$\Rightarrow x + (E - E)$
$\Rightarrow x + (V - E)$
$\Rightarrow x + (y - E)$
$\Rightarrow x + (y - V)$
$\Rightarrow x + (y - z)$

Figura 5.3.4. Árvore de derivação do exemplo 5.3.4.

5.3.2. Ambiguidade nas gramáticas e nas linguagens

Vimos que dada uma cadeia $w \in L(G)$, o parsing exaustivo produz uma árvore de derivação. Relembremos que uma árvore de derivação tem uma derivação pela esquerda e uma derivação pela direita, e portanto existem duas derivações para a mesma cadeia associadas à mesma árvore. Situação bem distinta é aquela em que existem diversas árvores de derivação para a mesma gramática. Neste caso há mais do que uma derivação pela esquerda e mais do que uma derivação pela direita. Qual usar? Temos uma situação de **ambiguidade** na linguagem, no sentido de que não se sabe que árvore usar.

Definição 5.3.3

Uma gramática livre de contexto é **ambígua** se existir alguma cadeia $w \in L(G)$ que tem pelo menos duas árvores de derivação possíveis. A ambiguidade implica a existência de duas ou mais derivações de extrema esquerda ou de extrema direita.

Exemplo 5.3.5(Linz) :

Seja a gramática com produções

$$S \rightarrow aSb \mid SS \mid \lambda$$

construir a árvore de parsing de *aabb*.

Há várias possibilidades:

i) $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

ii) $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aaSbbS \Rightarrow aabbS \Rightarrow aabb$

iii) $S \Rightarrow SS \Rightarrow SaSb \Rightarrow SaaSbb \Rightarrow Saabb \Rightarrow aabb$

A que correspondem as árvores seguintes, Fig. 5.3.5.

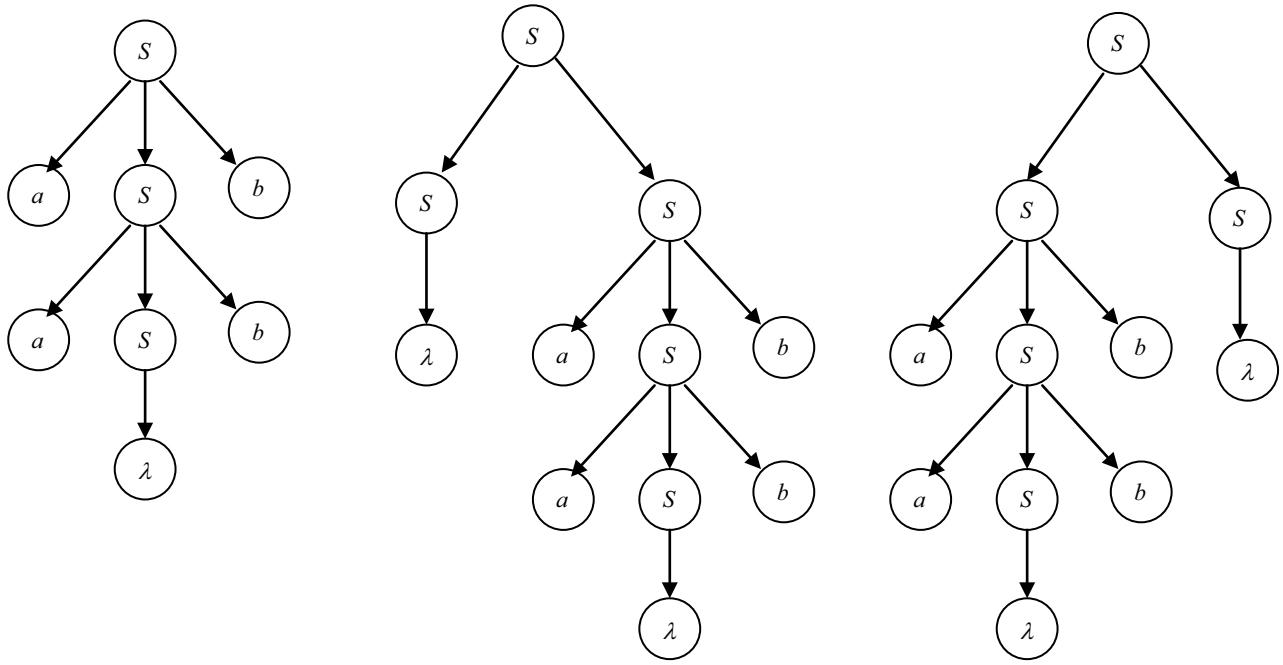


Figura 5.3.5. Árvores de derivação do exemplo 5.3.5

A segunda e terceira diferem apenas no facto de corresponderem a produções pela esquerda ou pela direita.

Contrariamente às linguagens naturais, em que a ambiguidade é tolerada (tendo até valor literário), nas linguagens de programação só pode existir uma interpretação para cada cadeia (sentença) e por isso não pode existir ambiguidade. Poderia acontecer uma tragédia se um compilador pudesse fazer o parsing de um programa de dois modos: resultariam dois códigos executáveis diferentes. Se uma gramática é ambígua, ela deve por isso ser rescrita para a libertar de toda a ambiguidade.

A ambiguidade pode estar na linguagem ela própria.

Exemplo 5.3.6

Seja a gramática com as produções

$$S \to AS \mid a \mid b$$

$A \to SS \mid ab$ Para gerar a cadeia *abb* podem-se seguir duas árvores de derivação, Fig. 5.3.6

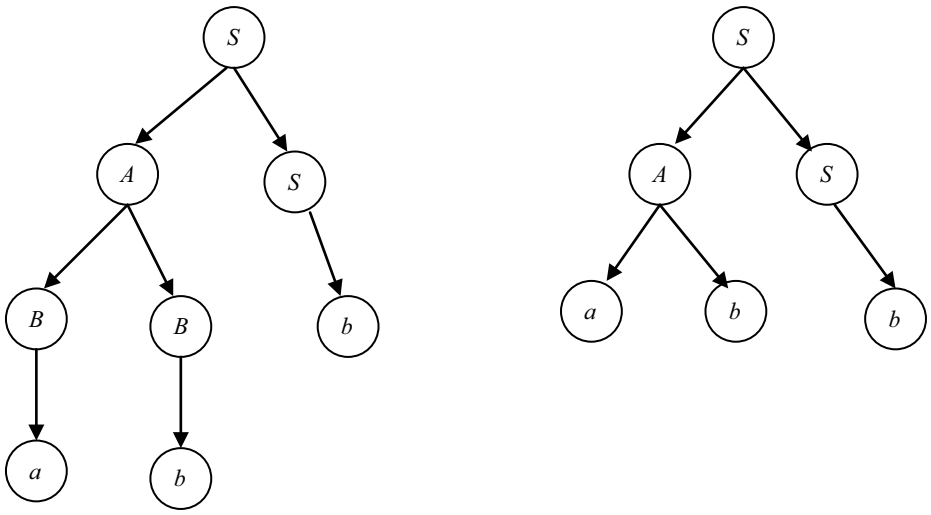


Figura 5.3.6. Exemplo 5.3.6.

Vejamos ainda outro exemplo (Linz, 142, Hopcroft 172).

Vamos desenvolver a gramática que produz cadeias do tipo $(a+b)*c$, $a+b*c$, $a+b+c$, ... Trata-se de uma gramática de expressões aritméticas, do tipo das usadas por linguagens de programação), em versão muito simplificada, considerando apenas os operadores adição + e multiplicação * . Os argumentos dos operadores são identificadores, e estes podem ser neste exemplo *a*, *b* ou *c*.

Necessitamos de duas variáveis nesta gramática. A variável *E* representa as expressões; é o símbolo inicial. A variável *I* representa os identificadores. Teremos assim a gramática $G= \{V, T, E, P\}$ com

$V=\{E, I\}$

$T=\{a, b, c, +, *, (,)\}$ e as produções

P: $P1.\quad E \rightarrow I$

$P2.\quad E \rightarrow E + E$

$P3\quad E \rightarrow E * E$

$P4\quad E \rightarrow (E)$

$P567\quad I \rightarrow a \mid b \mid c$

Para derivar a cadeia $a+b*c$ podem-se seguir duas árvores de derivação, Fig. 5.3.7

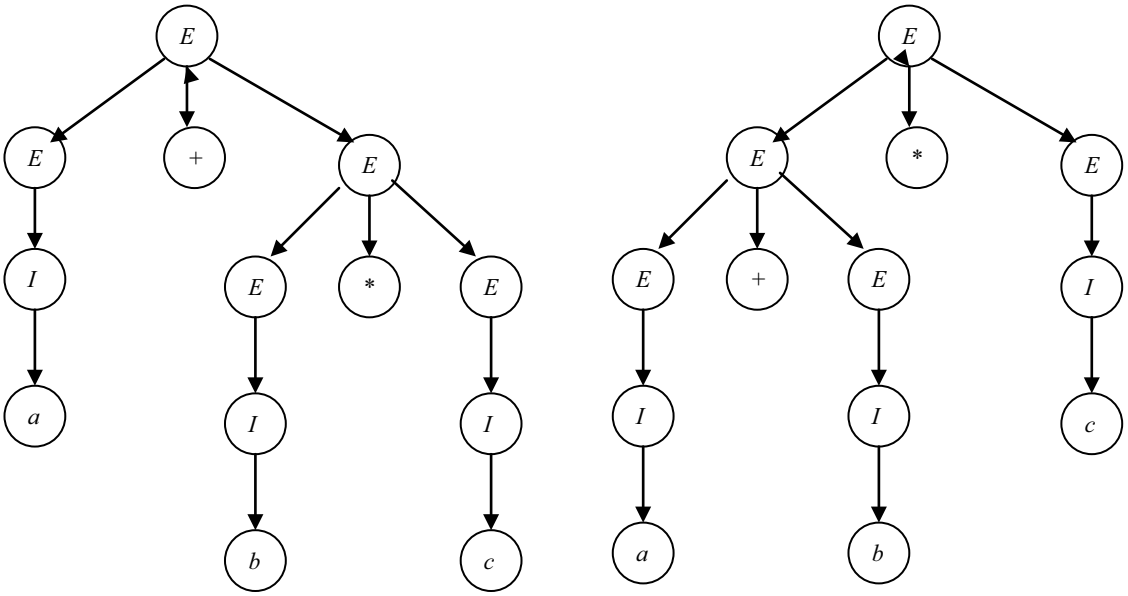


Figura 5.3.7. Árvores das expressões aritméticas

Na árvore da esquerda a subcadeia $b*c$ é filha de E, gerada depois da subcadeia $a+$. Na da direita $a+b$ é filha E, gerada antes de $*c$. Tendo em conta que a árvore de derivação define a estrutura de dados usada pelo compilador, teríamos na esquerda o código correspondente a *(a)+(b*c)* e na direita o código correspondente a $(a+b)*(c)$. Só a primeira está certa. Temos aqui o problema da precedência dos operadores aritméticos. Uma gramática assim poderia produzir resultados errados. A gramática é ambígua e é a sua ambiguidade que leva e esta problema.

Se o leitor desenhar a árvore de derivação de $a+b+c$ encontrará também duas soluções possíveis, uma correspondente a $a+(b+c)$ e outra a $(a+b)+c$. Neste caso não há problemas de compilação, mas há ambiguidade na mesma.

Para levantar a ambiguidade deve-se alterar a gramática.

Introduzam-se mais duas variáveis, fazendo

$$V=\{E, T, F, I\},$$

e façam-se as produções

$$\begin{array}{ll} P1. & E \rightarrow T \\ P2. & T \rightarrow F \\ P3. & F \rightarrow I \\ P3. & E \rightarrow E + T \\ P4 & T \rightarrow T^* F \\ P5 & F \rightarrow (E) \\ P678 & I \rightarrow a \mid b \mid c \end{array}$$

Agora para derivar a mesma cadeia $a+b*c$ teremos a árvore da Fig. 5.3.8.

O leitor pode tentar uma outra árvore de derivação desta cadeia. Não a encontrará porque a gramática é não ambígua. Note-se que este problema de saber se uma gramática é ou não ambígua não tem ainda uma solução geral; só para algumas gramáticas, por análise própria, se pode obter a resposta.

Esta gramática produz a mesma linguagem da gramática ambígua e nesse sentido as duas são equivalentes. No entanto podem produzir resultados de compilação diferentes.

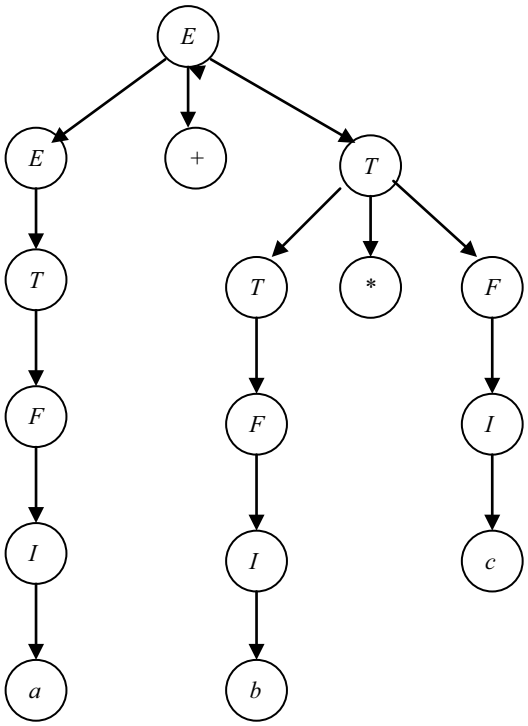


Figura 5.3.8.Árvore de derivação refeita

Exemplo 5.3.7

Se quisermos uma gramática não-ambígua de expressões aritméticas com identificadores mais gerais, podendo conter *a*'s, *b*'s, *c*'s, 0's e 1's, mas iniciando-se sempre por uma das letras (a linguagem dos identificadores é a da expressão regular $(a+b+c)(a+b+c+0+1)^*$), basta substituir na gramática anterior as produções *P678* por $I \rightarrow a \mid b|c| Ia \mid Ib|Ic \mid I0|I1|$, obtendo-se a gramática escrita em forma compacta seguinte (Hopcroft, 172):

$$E \rightarrow T \mid E+T$$

$$T \rightarrow F \mid T^*F$$

$$F \rightarrow I \mid (E)$$

$$I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1$$

Há gramáticas ambíguas que se podem tornar em gramáticas não ambíguas por pequenas alterações na sua estrutura, como foi o caso deste exemplo. Nem sempre assim acontece. De facto há linguagens que são elas mesmas ambíguas e por isso não é possível encontrar para elas uma gramática não ambígua.

Definição 5.3.4

Se L é uma linguagem livre de contexto para a qual existe uma gramática não ambígua, então L diz-se não ambígua. Se toda a gramática que gera L é ambígua, neste caso a linguagem diz-se **inerentemente ambígua.**

Exemplo de linguagem inerentemente ambígua (Hopcroft, 212)

$$L = \{a^n b^n c^m d^m, n \geq 1, m \geq 1) \cup \{a^n b^m c^m d^n, n \geq 1, m \geq 1\}$$

Esta linguagem é composta por todas as cadeias $a^+ b^+ c^+ d^+$ tal que

i) ou existem tantos a's e b's e tantos c's e d's

ii) ou existem tantos a's e d's e tantos b's e c's

Esta linguagem é livre de contexto.

Uma gramática para ela:

i) Para $a^n b^n c^m d^m$

$$S \rightarrow AB$$
$$A \rightarrow aAb \mid ab$$
$$B \rightarrow cBd \mid cd$$

ii) Para $a^n b^m c^m d^n$

$$S \rightarrow C$$
$$C \rightarrow aCd \mid aDd$$
$$D \rightarrow bDc \mid bc$$

Juntando agora as duas partes através da produção $S \rightarrow AB \mid C$ vem

$$S \rightarrow AB \mid C$$
$$A \rightarrow aAb \mid ab$$
$$B \rightarrow cBd \mid cd$$
$$C \rightarrow aCd \mid aDd$$
$$D \rightarrow bDc \mid bc$$

Esta gramática é ambígua. A cadeia *aabbccdd* pode gerar-se por duas derivações de extrema esquerda,

1. $S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcBd \Rightarrow aabbccdd$

2. $S \Rightarrow C \Rightarrow aCd \Rightarrow aaDdd \Rightarrow aabDcdd \Rightarrow aabbcdd$

cujas árvores de parsing são as seguintes Fig 5.3.9 3 5.3.10 (com folhas simplificadas):

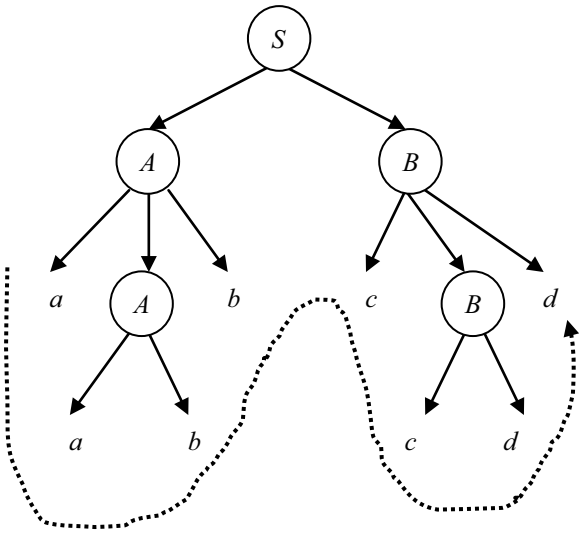


Figura 5.3.9

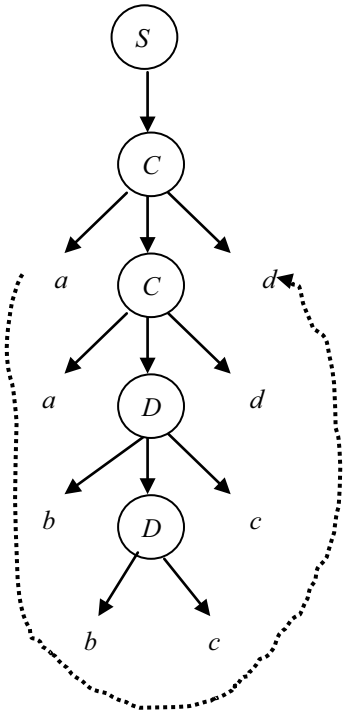


Figura 5.3.10

5.4. Gramáticas livres de contexto e linguagens de programação

As linguagens de programação, com por exemplo o Phyton, Java, o C, o Pascal, são definidas por gramáticas livres de contexto. Para o caso do Java existem diversas versões. A completa, desenvolvida pela Sun Microsystems, encontra-se na *Java Language Specification* em, http://java.sun.com/docs/books/jls/second_edition/html/jTOC.doc.html

Pode ver-se uma outra versão em http://www.lykkenborg.no/java/grammar/JLS3.html

Encontram-se também versões muito simplificadas, para fins didácticos, com o a de http://www.willamette.edu/~fruehr/231/grammar/simplest.html ou a especificação BNF[1] para uma mini Java em http://www.cambridge.org/resources/052182060X/MCIIJ2e/grammar.htm.

Para o Phyton ver uma gramática BNF em http://docs.python.org/ref/grammar.txt.

Bibliografia

An Introduction to Formal Languages and Automata, Peter Linz, 3rd Ed., Jones and Bartelett Computer Science, 2001

Introduction to Automata Theory, Languages and Computation, 2nd Ed., John Hopcroft, Rajeev Motwani, Jeffrey Ullman, Addison Wesley, 2001.

Elements for the Theory of Computation, Harry Lewis and Christos Papadimitriou, 2nd Ed., Prentice Hall, 1998.

Introduction to the Theory of Computation, Michael Sipser, PWS Publishing Co, 1997.

[1] BNF significa "Backus Naur Form".Para mais informação ver por exemplo http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html.