LINGUAGENS FORMAIS: Teoria e Conceitos

Material adicional, versão do dia 21 de abril de 2025 às 16:55

1. Página 15:
   Substituir "1.8 Conjuntos enumeráveis" por "1.8 Cardinalidade de conjuntos".

2. Página 20:
   Onde se lê (na última linha): de qualquer conjunto A.
   Leia-se: de qualquer conjunto A (veja Exemplo 1.56).

3. Página 20:
   Antes de "Os conjuntos $\emptyset$ e A ..." na penúltima linha, inserir:
   O símbolo $\not\subseteq$ é usado para indicar que um conjunto não é subconjunto de outro. Por exemplo, $A \not\subseteq B$ indica que A não é subconjunto de B.

4. Página 20:
   Substituir as duas últimas frases "Os conjuntos $\emptyset$ e A são, por definição, subconjuntos de qualquer conjunto A. Note que $\emptyset \subseteq \emptyset$" pelo parágrafo:
   "Os conjuntos $\emptyset$ e A são, por definição, subconjuntos de qualquer conjunto A. Para provar que o conjunto vazio é subconjunto de qualquer conjunto ($\forall A, \emptyset \subseteq A$) deve-se provar $\forall x, x \in \emptyset \Rightarrow x \in A$. Como, pela definição de $\emptyset$, a hipótese é falsa ($x \in \emptyset$), segue que a implicação é verdadeira. Outra maneira é por contradição, supondo que $\emptyset \not\subseteq A$. Se isto for verdade, então existe $x \in \emptyset$ tal que $x \notin A$. Como $x \in \emptyset$ é falso (pela definição de $\emptyset$), segue que a hipótese $\emptyset \not\subseteq A$ é falsa e portanto $\emptyset \subseteq A$. Para provar que todo conjunto é subconjunto dele mesmo ($\forall A, A \subseteq A$), basta mostrar que $\forall x, x \in A \Rightarrow x \in A$, o que é trivial (pois todo elemento de um conjunto é elemento deste conjunto). Note que $\emptyset \subseteq \emptyset$.

5. Página 21:
   Onde se lê:
   "ocorrendo, portanto, quando no máximo apenas uma das duas condições $M \subseteq N$ e $N \subseteq M$ for verdadeira".
   Leia-se:
   "ocorrendo, portanto, quando no máximo apenas uma das duas condições $M \subseteq N$ e $N \subseteq M$ for verdadeira ou, ainda, de forma equivalente, quando $M \not\subseteq N$ ou $N \not\subseteq M$."

6. Página 21:
   Inserir logo depois de "verdadeira." no segundo parágrafo:
   Em outras palavras, $M = N \Leftrightarrow (M \subseteq N) \wedge (N \subseteq M)$ e $M \neq N \Leftrightarrow \overline{(M \subseteq N) \wedge (N \subseteq M)} = \overline{(M \subseteq N)} \vee \overline{(N \subseteq M)} = (M \not\subseteq N) \vee (N \not\subseteq M)$.

7. Página 25:
   Inserir no início do parágrafo "No caso da relação (ii)..."
   Note que a relação (i) engloba o caso em que $A \subseteq B$ ou $B \subseteq A$.

8. Página 44:
   Substituir a primeira frase da Seção 1.7 por:
   Linguagens formais e autômatos constituem sistemas matemáticos nos quais inúmeras propriedades, em geral formuladas como proposições na forma **teoremas**, **lemas** ou **corolários**, podem ser inferidas a partir de verdades previamente conhecidas ou admitidas por hipótese, por intermédio de raciocínios lógicos expressos como **demonstrações** (ou **provas**).

9. Página 44:
   Inserir, antes de **Provas envolvendo conjuntos**:
   Uma excelente referência introdutória sobre as diversas maneiras de se obter a prova de proposições envolvendo os diversos conectivos lógicos, conjuntos e números inteiros, é citar loehr2019.

10. Página 44:
    Onde se lê: $x \in A \Rightarrow x \in B$
    Leia-se: $\forall x, x \in A \Rightarrow x \in B$

11. Página 45:
    Onde se lê: $x \in B \Rightarrow x \in A$
    Leia-se: $\forall x, x \in B \Rightarrow x \in A$

12. Página 45:
    Depois de: Provar que todo elemento de A é também elemento de B.
    Acrescentar: $(\forall x, x \in A \Rightarrow x \in B)$

13. Página 45:
    Depois de: Provar que existe (pelo menos um) elemento de B que não é elemento de A.
    Acrescentar: $(\exists x, x \in B \wedge x \notin A)$

14. Página 51:
    Seção 1.8:
    Acrescentar, no final do parágrafo "Trata-se de...":
    Se A possuir uma quantidade de elementos que é maior ou igual que a de B, então $| A | \geq | B |$.

15. Página 51:
    Seção 1.8:
    Substituir o Exemplo 1.67 por:
    Exemplo 1.67 Considerem-se os conjuntos finitos $A = \{a,b,c,d\}$, $B = \{0,1,2,3,4,5\}$ e $C = \{s,t,u,v\}$. Então, $| A | = 4$, $| B | = 6$, $| C | = 4$, $| A | \leq | B |$, $| A | < | B |$, $| A | = | C |$, $| B | \geq | C |$ e $| B | > | C |$.

16. Página 51:
    Seção 1.8:
    Substituir do parágrafo "De que forma ..." (inclusive) até o final da página por:
    De que forma seria, por outro lado, possível comparar o "tamanho" de dois conjuntos infinitos? Assim como no caso dos conjuntos finitos, dois conjuntos infinitos também possuem cardinalidades relativas. Para caracterizar a natureza desta relação, basta identificar (ou provar que não existem) certas funções entre os conjuntos.

    Note, no caso dos conjuntos finitos, que as noções acima (envolvendo a quantidade de elementos) podem ser substituídas por *existe uma bijeção* (para a relação "=" entre a cardinalidade de conjuntos), *existe uma função injetora e total* (para a relação "$\leq$" entre a cardinalidade de conjuntos) e *existe uma função injetora e total e não existe uma bijeção* (para a relação "$<$" entre a cardinalidade de conjuntos).

    Exemplo 1.68 Sejam $A = \{a,b,c\}$ e $B = \{7,3,6\}$. Neste exemplo, A e B possuem a mesma cardinalidade, pois $|A| = |B| = 3$. Note-se que é possível definir uma função bijetora de A para B: $\{(a,7), (b,3), (c,6)\}$. Naturalmente, muitas outras funções bijetoras também podem ser definidas entre esses dois conjuntos.

    Exemplo 1.69 Sejam $A = \{a \mid a \text{ é ímpar}, 1 \leq a \leq 100\}$ e $B = \{b \mid b \text{ é par}, 1 \leq b \leq 100\}$. A e B são conjuntos finitos que possuem a mesma cardinalidade, pois a função $f(a) = a + 1$ é bijetora, mapeando os elementos do conjunto A nos elementos do conjunto B. Neste caso, $|A| = |B| = 50$.

    Exemplo 1.70 Sejam $A = \{a,b,c\}$ e $B = \{1,2,3,4,5\}$. Como $| A | = 3$ e $| B | = 5$, segue que $| A | \leq | B |$. De fato, uma função injetora entre A e B é, entre outras, $\{(a,1),(b,2),(c,3)\}$. Por outro lado, a inexistência de uma bijeção entre A e B implica em $| A | < | B |$.

    Como não é possível estabelecer a cardinalidade relativa de conjuntos infinitos pelo mesmo critério usado para os conjuntos finitos, serão então usadas as noções abaixo, que são válidas para ambos os casos. Sejam A e B dois conjuntos (ambos finitos ou ambos infinitos). A cardinalidade relativa de A e B é portanto:

    $=$ Se houver uma bijeção entre A e B.

    $\leq$ Se houver uma função injetora e total entre A e B.

    $<$ Se houver uma função injetora e total entre A e B e não houver uma bijeção entre A e B.

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

Exemplo 1.71 Considere-se o conjunto dos números inteiros $\mathbb{Z}$ e o subconjunto de $\mathbb{Z}$ composto apenas pelos números ímpares. Trata-se, naturalmente, de dois conjuntos infinitos, sendo o segundo um subconjunto próprio do primeiro. Porém, de acordo com a definição, embora isso pareça paradoxal, os dois conjuntos possuem a mesma cardinalidade, já que a função bijetora $2 * i + 1$, onde $i \in \mathbb{Z}$, mapeia univocamente cada elemento de $\mathbb{Z}$ em um único elemento do conjunto dos números ímpares.

Do Exemplo 1.71 pode-se observar facilmente que, diferentemente do que ocorre com conjuntos finitos, é possível, para conjuntos infinitos, definir subconjuntos próprios com a mesma cardinalidade do conjunto original. Em outras palavras, suponha que A e B sejam conjuntos finitos. Então, se $| A | = | B |$, A e B necessariamente possuem a mesma quantidade de elementos. Se A possuir mais elementos do que B, então $| A | > | B |$. Suponha agora que A e B sejam conjuntos infinitos. Então, se $| A | = | B |$, A e B podem ou não possuir a mesma quantidade de elementos. Ou seja, o fato de um conjunto infinito possuir mais elementos do que outro conjunto infinito não garante que as suas respectivas cardinalidades sejam diferentes (exemplo: $\mathbb{Z}$ possui mais elementos do que $\mathbb{N}$ mas ambos possuem a mesma cardinalidade; $\mathbb{R}$ possui mais elementos do que $\mathbb{N}$ e a cardinalidade do primeiro é maior do que a do segundo). Se A é finito e B é infinito, então necessariamente $| A | < | B |$.

17. `Página 59:`
    `Inserir, como último parágrafo da Seção 1.8:`
    Qual a relação que existe entre quantidade de elementos e cardinalidade? Em resumo, suponha que A e B sejam dois conjuntos com quantidades diferentes de elementos. Se A e B forem ambos finitos, então a cardinalidade deles será necessariamente diferente (a cardinalidade maior será a do conjunto com a maior quantidade de elementos). Dois conjuntos finitos só possuem a mesma cardinalidade se ambos possuirem a mesma quantidade de elementos. Se um conjunto for finito e o outro infinito, então a cardinalidade do conjunto infinito será sempre maior do que a cardinalidade do conjunto finito (uma vez que é possível estabelecer uma função total e injetora entre ambos, mas é impossível estabelecer uma bijeção entre os dois). Mas se A e B forem ambos infinitos, então a quantidade de elementos não informa nada sobre a cardinalidade relativa destes conjuntos. Suponha $\mathbb{N}$, $\mathbb{Z}$ e $\mathbb{R}$. Os três são infinitos e possuem quantidades diferentes de elementos ($\mathbb{N} \subset \mathbb{Z} \subset \mathbb{R}$). No entanto, conforme visto anteriormente, $| \mathbb{N} | = | \mathbb{Z} |$ e $| \mathbb{N} | < | \mathbb{R} |$. Desta maneira, a comparação da cardinalidade de dois conjuntos infinitos só pode ser feita a partir da prova da existência ou inexistência de uma bijeção entre ambos e também da existência ou inexistência de uma função total e injetora ente ambos.

18. `Página 65:`
    `Acrescentar ao final do segundo párágrafo:`
    Note que $| \alpha \cdot \beta | = | \alpha | + | \beta |$.

19. `Página 65:`
    `Acrescentar depois de` $\varepsilon^R = \varepsilon$
    A operação reverso é **idempotente**, ou seja, $(\alpha^R)^R = \alpha$ para qualquer α.

20. Página 66:
    `Novo parágrafo antes da Seção 2.2 Linguagens`
    Se α é uma **cadeia de cadeias** sobre o alfabeto Σ, então:

$$\alpha = \mu_1\mu_2...\mu_{n-1}\mu_n, \text{ com } \mu_i \in \Sigma^*, 1 \leq i \leq n, \text{ isto implica } \alpha^R = \mu_n^R\mu_{n-1}^R...\mu_2^R\mu_1^R$$

    Este resultado não será provado.

21. `Página 69:`
    `Acrescentar, logo depois do Exemplo 2.13:`
    Note que a operação de concatenação é distributiva em relação à operação de união, tanto à esquerda quanto à direita:

$$X(Y \cup Z) = XY \cup XZ$$
$$(X \cup Y)Z = XZ \cup YZ$$

22. Página 80:
Acrescentar, antes do Exemplo 2.31:
Uma gramática $G = (V, \Sigma, P, S)$ é dita "bem formada" se e somente se:

- $\Sigma \subset V$.
- Σ é finito e não-vazio.
- $N = V - \Sigma$ é finito e não-vazio.
- $S \in N$.
- Se $\alpha \to \beta \in P$, então $\alpha \in V^*NV^*$ e $\beta \in V^*$.

Caso contrário, diz-se que G está "malformada".

23. Página 82:
Acrescentar, antes do parágrafo "A completa identificação...":
Para que uma gramática G defina uma linguagem L, as duas condições seguintes devem ser observadas simultaneamente:

- Toda sentença de L pode ser gerada por G.
- Toda cadeia não pertencente à L não pode ser gerada por G (ou seja, não existe nenhuma cadeia não pertencente à L que possa ser derivada em G).

Exemplo 2.34 Considere a linguagem L formada por todas as cadeias sobre o alfabeto $\{a,b\}$ que começam com a e terminam com b. A gramática G_i com as regras $S \to aXb, X \to aX, X \to bX, X \to a, X \to b$ não gera L pois falha em gerar a sentença ab, pertencente à ela (viola a primeira condição acima). Ou seja, $L(G_i) = L - \{ab\}$. Por outro lado, a gramática G_{ii} com as regras $S \to aXb, S \to a, X \to aX, X \to bX, X \to \varepsilon$ não gera L pois permite a derivação da cadeia a, que não pertence à L (viola a segunda condição acima). Ou seja, $L(G_{ii}) = L \cup \{a\}$. A gramática G_{iii} com as regras $S \to aXb, X \to aX, X \to bX, X \to \varepsilon$ gera exatamente L, pois ambas as condições acima são verificadas.

Note que a escolha do não-terminal inicial (raiz da gramática) é determinante para a formação da linguagem gerada pela respectiva gramática.

Exemplo 2.35 Considere novamente o Exemplo 2.33. Se a raiz de G_1 fosse A e não S_1, então $L(G_1) = \{\varepsilon, 12\}$. Ou seja, a linguagem seria finita e composta por duas sentenças, uma de comprimento zero (ε) e outra de comprimento dois (12). O símbolo não-terminal S_1, neste caso, não teria nenhuma utilidade na gramática e na linguagem gerada por ela.

24. Página 82:
Parágrafo "A completa identificação...":
onde se lë:
" Observe-se, a título de ilustração, o caso da gramática G_2 apresentada no Exemplo 2.34."
leia-se:
" Observe-se, a título de ilustração, os casos das gramáticas G_2 e G_3 apresentadas, respectivamente, nos Exemplos 2.36 e 2.37."

25. Página 82:
Acrescentar, antes do Exemplo 2.34:
Exemplo 2.36 A gramática $G_2 = (\{a,b,S,X\}, \{a,b\}, \{S \to aSb, S \to X, X \to ab\}, S)$ é tal que $L(G_2) = \{a^n b^n$ com $n \geq 1\}$, portanto infinita. Trata-se de uma gramática simples cuja respectiva linguagem gerada não é difícil de ser deduzida. De fato, $S \Rightarrow^i a^i S b^i \Rightarrow a^i X b^i \Rightarrow a^i abb^i = a^{i+1} b^{i+1}$.

Uma pequena mudança na linguagem pode introduzir uma grande complexidade na correspondente gramática. Veja o Exemplo 2.37, que apresenta uma gramática que gera a linguagem $a^n b^n c^n$ com $n \geq 1$.

26. Página 82:
Exemplo 2.34:
Substituir *Exemplo* 2.34 por *Exemplo* 2.37
substituir G_2 por G_3, V_2 por V_3, Σ_2 por Σ_3 e P_2 por P_3

27. Página 83:
Exemplo 2.35:
Substituir *Exemplo* 2.35 por *Exemplo* 2.38
substituir G_3 por G_4 e G_4 por G_5

28. Página 94:
Acrescentar, antes do parágrafo "É possível...":
Diz-se que um reconhecedor R define uma linguagem L quando:

- Todas as sentenças de L são aceitas por R, e
- Todas as cadeias não pertencentes à L são rejeitadas por R (ou seja, não existe nenhuma cadeia não pertencente à L que seja aceita por R).

29. Página 110:
Acrescentar, antes do Exemplo 3.1:
ou seja, $\alpha \in N$ e $\beta \in \{\varepsilon\} \cup \Sigma \cup N \cup \Sigma N$. Logo, $|\alpha| = 1$ e $|\beta| \leq 2$.

30. Página 110:
Acrescentar, antes do Exemplo 3.2:
ou seja, $\alpha \in N$ e $\beta \in \{\varepsilon\} \cup \Sigma \cup N \cup N\Sigma$. Logo, $|\alpha| = 1$ e $|\beta| \leq 2$.

31. Página 113:
Acrescentar, antes do último parágrafo "Se, na GLD, as ..."
Prova:

Na Seção 1.6 introduzimos o símbolo "$\Rightarrow$" como sendo a representação da implicação lógica. Depois, na Seção 2.3, o mesmo símbolo foi reutlizado para denotar a obtenção de uma forma sentencial a partir de outra em gramáticas (através de derivações). No que segue, o símbolo será usado com ambos os significados e o entendimento deverá ser feito em função do contxto. Deseja-se provar:

$$(S \Rightarrow^*_{G_1} w) \Leftrightarrow (S \Rightarrow^*_{G_2} w)$$

ou seja, que toda sentença derivável na GLD G_1 pode ser também ser derivada na GLE G_2 e vice-versa. Esta prova, portanto, se desdobra em duas:

(a) $(S \Rightarrow^*_{G_1} w) \Rightarrow (S \Rightarrow^*_{G_2} w)$
(b) $(S \Rightarrow^*_{G_1} w) \Leftarrow (S \Rightarrow^*_{G_2} w)$

Para o primeiro caso, a obtenção de G_2 a partir de G_1 é feita conforme o Algoritmo 3.2 A prova é feita por indução na quantidade de derivações n:

- Caso base:
  $(n = 1)$:
  $(S \Rightarrow^1_{G_1} w) \Rightarrow (S \Rightarrow^1_{G_2} w)$
  De fato, se $S \Rightarrow^1_{G_1} w$, então existe uma regra $S \to \mu \in P$ tal que $S \Rightarrow^1_{G_1} \mu$ e $w = \mu$. Como esta mesma regra existe em P' (de G_2), segue que $S \Rightarrow^1_{G_2} \mu$ e $w = \mu$.
- Caso indutivo:
  $(n \Rightarrow n+1)$:
  $((S \Rightarrow^n_{G_1} w) \Rightarrow (S \Rightarrow^n_{G_2} w)) \Rightarrow ((S \Rightarrow^{n+1}_{G_1} w) \Rightarrow (S \Rightarrow^{n+1}_{G_2} w))$ Podemos, portanto, assumir como hipóteses:
  - $((S \Rightarrow^n_{G_1} w) \Rightarrow (S \Rightarrow^n_{G_2} w))$ e que
  - $(S \Rightarrow^{n+1}_{G_1} w)$

  O detalhamento da primeira sequência de derivações $(S \Rightarrow^n_{G_1} w)$ revela:

$$\begin{aligned} S &\Rightarrow_{G_1} \mu_1 A_1 \\ &\Rightarrow_{G_1} \mu_1\mu_2 A_2 \\ &\Rightarrow^*_{G_1} \mu_1\mu_2...\mu_{n-2}A_{n-2} \\ &\Rightarrow_{G_1} \mu_1\mu_2...\mu_{n-2}\mu_{n-1}A_{n-1} \\ &\Rightarrow_{G_1} \mu_1\mu_2...\mu_{n-2}\mu_{n-1}\mu_n \end{aligned}$$

com $w = \mu_1\mu_2\mu_3...\mu_{n-1}\mu_n$. De fato, as seguintes regras de G_1 foram usadas nesta derivação:

- $S \to \mu_1 A_1$
- $A_1 \to \mu_2 A_2$
- ...
- $A_{n-2} \to \mu_{n-1}A_{n-1}$
- $A_{n-1} \to \mu_n$

Conforme o Algoritmo 3.2, as seguintes regras devem ser usadas em G_2 (na ordem inversa):

- $A_1 \to \mu_1$
- $A_2 \to A_1\mu_2$
- ...
- $A_{n-1} \to A_{n-2}\mu_{n-1}$
- $S \to A_{n-1}\mu_n$

Portanto (como antecipado pela hipótese indutiva):

$$\begin{aligned} S &\Rightarrow_{G_2} A_{n-1}\mu_n \\ &\Rightarrow_{G_2} A_{n-2}\mu_{n-1}\mu_n \\ &\Rightarrow^*_{G_2} A_2...\mu_{n-1}\mu_n \\ &\Rightarrow_{G_2} A_1\mu_2...\mu_{n-1}\mu_n \\ &\Rightarrow_{G_2} \mu_1\mu_2...\mu_{n-1}\mu_n \end{aligned}$$

com $w = \mu_1\mu_2\mu_3...\mu_{n-1}\mu_n$. Devemos, agora, provar que $S \Rightarrow^{n+1}_{G_2} w$. Para excutar $n+1$ derivações em $S \Rightarrow^{n+1}_{G_1} w$, basta modificar a última regra usada e, depois dessa, usar mais uma:

- $S \to \mu_1 A_1$
- $A_1 \to \mu_2 A_2$
- ...
- $A_{n-2} \to \mu_{n-1}A_{n-1}$
- $\boxed{A_{n-1} \to \mu_n A_n}$ no lugar de $A_{n-1} \to \mu_n$
- $\boxed{A_n \to \mu_{n+1}}$

ou seja,

$$\begin{aligned} S &\Rightarrow_{G_1} \mu_1 A_1 \\ &\Rightarrow_{G_1} \mu_1\mu_2 A_2 \\ &\Rightarrow^*_{G_1} \mu_1\mu_2...\mu_{n-2}A_{n-2} \\ &\Rightarrow_{G_1} \mu_1\mu_2...\mu_{n-1}\mu_{n-1}A_{n-1}n \\ &\Rightarrow_{G_1} \mu_1\mu_2...\mu_{n-1}\mu_n A_n \\ &\Rightarrow_{G_1} \mu_1\mu_2...\mu_{n-1}\mu_n\mu_{n+1} \end{aligned}$$

com $w = \mu_1\mu_2\mu_3...\mu_{n-1}\mu_n\mu_{n+1}$. Conforme o Algoritmo 3.2, as seguintes regras devem ser usadas em G_2 (na ordem inversa):

- $A_1 \rightarrow \mu_1$
- $A_2 \rightarrow A_1\mu_2$
- ...
- $A_{n-1} \rightarrow A_{n-2}\mu_{n-1}$
- $\boxed{A_n \rightarrow A_{n-1}\mu_n}$
- $\boxed{S \rightarrow A_n\mu_{n+1}}$

ou seja,

$$\begin{aligned} S &\Rightarrow_{G_2} A_n\mu_{n+1} \\ &\Rightarrow_{G_2} A_{n-1}\mu_n\mu_{n+1} \\ &\Rightarrow^*_{G_2} A_2...\mu_{n-1}\mu_n\mu_{n+1} \\ &\Rightarrow_{G_2} A_1\mu_2...\mu_{n-1}\mu_n\mu_{n+1} \\ &\Rightarrow_{G_2} \mu_1\mu_2...\mu_{n-1}\mu_n\mu_{n+1} \end{aligned}$$

com $w = \mu_1\mu_2...\mu_{n-1}\mu_n\mu_{n+1}$. Com isso, concluimos a prova da parte 1 $((S \Rightarrow^*_{G_1} w) \Rightarrow (S \Rightarrow^*_{G_2} w))$. A parte 2 $((S \Rightarrow^*_{G_1} w) \Leftarrow (S \Rightarrow^*_{G_2} w))$ é similar e fica como exercício.

32. `Página 114:`
`Substituir o parágrafo "A constatação..." por:`
As regras da GLD G_1 empregadas na derivação anterior são as seguintes, por ordem de uso:

$$\begin{gathered} S \rightarrow \mu_nA_1 \\ A_1 \rightarrow \mu_{n-1}A_2 \\ A_2 \rightarrow \mu_{n-2}A_3 \\ A_3 \rightarrow \mu_{n-3}A_4 \\ ... \\ A_{n-2} \rightarrow \mu_2A_{n-1} \\ A_{n-1} \rightarrow \mu_1 \end{gathered}$$

Por outro lado, a Tabela 3.1 mostra que, para cada uma das regras acima da GLD G_1, existe uma regra correspondente na GLE G_2, a saber:

Regra da GLD G_1	Regra da GLE G_2
$S \rightarrow \mu_nA_1$	$A_1 \rightarrow \mu_n$
$A_1 \rightarrow \mu_{n-1}A_2$	$A_2 \rightarrow A_1\mu_{n-1}$
$A_2 \rightarrow \mu_{n-2}A_3$	$A_3 \rightarrow A_2\mu_{n-2}$
$A_3 \rightarrow \mu_{n-3}A_4$	$A_4 \rightarrow A_3\mu_{n-3}$
...	...
$A_{n-3} \rightarrow \mu_3A_{n-2}$	$A_{n-2} \rightarrow A_{n-3}\mu_3$
$A_{n-2} \rightarrow \mu_2A_{n-1}$	$A_{n-1} \rightarrow A_{n-2}\mu_2$
$A_{n-1} \rightarrow \mu_1$	$S \rightarrow A_{n-1}\mu_1$

Finalmente, a aplicação destas regras da GLD G_2, porém na ordem inversa, garante que a sentença derivada na GLD G_1 possa ser derivada na GLE G_2. Em outras palavras, temos que $L(G_1) \subseteq L(G_2)$.

33. `Página 115:`
`Substituir "A Tabela 3.1" por:`
De forma similar ao caso anterior, a Tabela 3.1

34. Página 116:
`Acrescentar, antes do último parágrafo` "Com G'' assim construída ..."
Prova:

Deseja-se provar:

$$(S \Rightarrow^*_{G'} w) \Leftrightarrow (S \Rightarrow^*_{G''} w^R)$$

ou seja, que toda sentença derivável na GLD G' pode ser também ser derivada (de forma reversa) na GLE G'' e vice-versa. Esta prova, portanto, se desdobra em duas:

(a) $(S \Rightarrow^*_{G'} w) \Rightarrow (S \Rightarrow^*_{G''} w^R)$
(b) $(S \Rightarrow^*_{G'} w) \Leftarrow (S \Rightarrow^*_{G''} w^R)$

Para o primeiro caso, a obtenção de G'' a partir de G' é feita conforme o Algoritmo 3.3 A prova é feita por indução na quantidade de derivações n:

- Caso base:
  $(n = 1)$:
  $(S \Rightarrow^1_{G'} w) \Rightarrow (S \Rightarrow^1_{G''} w^R)$
  De fato, se $S \Rightarrow^1_{G'} w$, então existe uma regra $S \to \mu \in P'$ tal que $S \Rightarrow^1_{G'} \mu$ e $w = \mu$. Como a regra $S \to \mu^R$ existe em P'' (de G''), segue que $S \Rightarrow^1_{G''} \mu^R = w^R$.
- Caso indutivo:
  $(n \Rightarrow n+1)$:
  $((S \Rightarrow^n_{G'} w) \Rightarrow (S \Rightarrow^n_{G''} w^R)) \Rightarrow ((S \Rightarrow^{n+1}_{G'} w) \Rightarrow (S \Rightarrow^{n+1}_{G''} w^R))$ Podemos, portanto, assumir como hipóteses:
  - $((S \Rightarrow^n_{G'} w) \Rightarrow (S \Rightarrow^n_{G''} w^R))$ e que
  - $(S \Rightarrow^{n+1}_{G'} w)$

  O detalhamento da primeira sequência de derivações $(S \Rightarrow^n_{G'} w)$ revela:

$$\begin{array}{lll} S & \Rightarrow_{G'} & \mu_1 A_1 \\ & \Rightarrow_{G'} & \mu_1 \mu_2 A_2 \\ & \Rightarrow^*_{G'} & \mu_1 \mu_2 ... \mu_{n-2} A_{n-2} \\ & \Rightarrow_{G'} & \mu_1 \mu_2 ... \mu_{n-2} \mu_{n-1} A_{n-1} \\ & \Rightarrow_{G'} & \mu_1 \mu_2 ... \mu_{n-1} \mu_n \end{array}$$

  com $w = \mu_1 \mu_2 \mu_3 ... \mu_{n-1} \mu_n$. De fato, as seguintes regras de G_1 foram usadas nesta derivação:
  - $S \to \mu_1 A_1$
  - $A_1 \to \mu_2 A_2$
  - ...
  - $A_{n-2} \to \mu_{n-1} A_{n-1}$
  - $A_{n-1} \to \mu_n$

  Conforme o Algoritmo 3.3, as seguintes regras devem ser usadas em G'' (na ordem direta):
  - $S \to A_1 \mu_1^R$
  - $A_1 \to A_2 \mu_2^R$
  - ...
  - $A_{n-2} \to A_{n-1} \mu_{n-1}^R$
  - $A_{n-1} \to \mu_n^R$

  Portanto (como antecipado pela hipótese indutiva):

$$
\begin{array}{rll}
S & \Rightarrow_{G''} & A_1\mu_1^R \\
& \Rightarrow_{G''} & A_2\mu_2^R\mu_1^R \\
& \Rightarrow^*_{G''} & A_{n-2}...\mu_2^R\mu_1^R \\
& \Rightarrow_{G''} & A_{n-1}\mu_{n-1}^R\mu_{n-2}^R...\mu_2^R\mu_1^R \\
& \Rightarrow_{G''} & \mu_n^R\mu_{n-1}^R\mu_{n-2}^R...\mu_2^R\mu_1^R \\
& = & (\mu_1\mu_2...\mu_{n-2}\mu_{n-1}\mu_n)^R \\
& = & w^R
\end{array}
$$

Devemos, agora, provar que $S \Rightarrow^{n+1}_{G''} w$. Para excutar $n+1$ derivações em $S \Rightarrow^{n+1}_{G'} w$, basta modificar a última regra usada e, depois dessa, usar mais uma:

- $S \to \mu_1 A_1$
- $A_1 \to \mu_2 A_2$
- ...
- $A_{n-2} \to \mu_{n-1} A_{n-1}$
- $\boxed{A_{n-1} \to \mu_n A_n}$ no lugar de $A_{n-1} \to \mu_n$
- $\boxed{A_n \to \mu_{n+1}}$

ou seja,

$$
\begin{array}{rll}
S & \Rightarrow_{G'} & \mu_1 A_1 \\
& \Rightarrow_{G'} & \mu_1\mu_2 A_2 \\
& \Rightarrow^*_{G'} & \mu_1\mu_2...\mu_{n-1}A_{n-1} \\
& \Rightarrow_{G'} & \mu_1\mu_2...\mu_{n-1}\mu_n A_n \\
& \Rightarrow_{G'} & \mu_1\mu_2...\mu_{n-1}\mu_n\mu_{n+1}
\end{array}
$$

com $\mu_1\mu_2...\mu_{n-1}\mu_n\mu_{n+1} = w\mu_{n+1}$. Conforme o Algoritmo 3.3, as seguintes regras devem ser usadas em G_2 (na ordem direta):

- $S \to A_1\mu_1^R$
- $A_1 \to A_2\mu_2^R$
- ...
- $A_{n-2} \to A_{n-1}\mu_{n-1}^R$
- $\boxed{A_{n-1} \to A_n\mu_n^R}$
- $\boxed{A_n \to \mu_{n+1}^R}$

ou seja,

$$
\begin{array}{rll}
S & \Rightarrow_{G''} & A_1\mu_1^R \\
& \Rightarrow_{G''} & A_2\mu_2^R\mu_n^R \\
& \Rightarrow^*_{G''} & A_{n-2}\mu_{n-2}^R...\mu_2^R\mu_1^R \\
& \Rightarrow_{G''} & A_{n-1}\mu_{n-2}^R...\mu_2^R\mu_1^R \\
& \Rightarrow_{G''} & A_n\mu_1^R\mu_{n-1}^R\mu_{n-2}^R...\mu_{n-1}^R\mu_n^R \\
& \Rightarrow_{G''} & \mu_{n+1}^R\mu_n^R\mu_{n-1}^R\mu_{n-2}^R...\mu_2 R\mu_1^R
\end{array}
$$

com $\mu^R_{n+1}\mu^R_n\mu^R_{n-1}\mu^R_{n-2}...\mu^R_2\mu^R_1 = \mu^R_{n+1}w^R = (w\mu_{n+1})^R$. Com isso, concluimos a prova da parte 1 ($(S \Rightarrow^*_{G'} w) \Rightarrow (S \Rightarrow^*_{G''})w^R$). A parte 2 ($(S \Rightarrow^*_{G'} w) \Leftarrow (S \Rightarrow^*_{G''})w^R$) é similar e fica como exercício.

35. Página 136:
Inserir, como novo parágrafo depois de "A Figura 3.11 ilustra o autômato modificado." :

O estado q_jq_k é criado com objetivo de representar simultaneamente os estados q_j e q_k. Assim, tal estado pode ser batizado com os nomes q_jq_k ou q_kq_j, pois a ordem dos nomes dos estados originais é irrelevante. Em outras palavras, pode-se dizer que q_jq_k (ou q_kq_j) representa o conjunto $\{q_j, q_k\}$. No caso de ser necessária a criação de um novo estado que represente, digamos, três outros estados (por exemplo q_r, q_s e q_t), então novo estado poderia receber os nomes $q_rq_sq_t$, $q_rq_tq_s$, $q_sq_rq_t$, $q_sq_tq_r$, $q_tq_rq_s$ ou $q_tq_sq_r$ — em qualquer caso, esse novo estado seria uma representação de $\{q_r, q_s, q_t\}$.

Além disso, é importante justificar o fato de que as transições que partem de q_j e q_k devem todas ser copiadas para q_jq_k: no autômato original (Figura 3.10), se o estado corrente for q_i e o simbolo corrente for a, então este autômato pode evoluir tanto para q_j quanto para q_k (em função do não determinismo) podendo, portanto, em seguida evoluir com as transições de q_j ou de que q_k. No autômato modificado (Figura 3.11), para se obter um efeito semelhante, deve-se permitir que, após a transição com o símbolo a, que este evolua conforme as transições de q_j ou de q_k ou, seja, com as transições de q_jq_k. Por este motivo as transições que partem de q_jq_k devem corresponder à união de todas as transições que partem de q_j com todas as transições que partem de q_k (mudando a origem para q_jq_k e preservando-se os destinos originais.

Criar um novo parágrafo a partir de "Fica claro também que..."

36. Página 174:
Inserir, antes do prágrafo "Em resumo:"
Em outras palavras: antes da execução do passo 2 do Algoritmo 3.14, as equações das variáveis X_i (no lado esquerdo) podem possuir X_j, $1 \leq j \leq m$ (ou seja, qualquer variável) do lado direito. Quando o passo 2 é executado sobre a equação da variável X_i, esta passa a conter, do lado direito, apenas referências à X_j, $j \geq i$. Ao término do passo 2, todas as equações de X_i contém referências apenas para X_j, $j \geq i$. Portanto, a última equação (de X_m, que contém apenas referências ao próprio X_m, no pior caso) é resolvida (por meio do Teorema 3.13) e tem início o passo 4 do algoritmo. Quando a solução de um certo X_i (obtida pelo Teorema 3.13) é substituida em todas as equações anteriores (das variáveis X_j, $j < i$), estas equações passam a conter, do lado direito, apenas a própria variável que está sendo definida. Uma equação com, por exemplo X_i do lado esquerdo, conterá apenas X_i do lado direito. Para resolver esta equação, basta usar novamente o Teorema 3.13.

37. Página 181:
Substituir, na penúltima linha, "autômato finito M" por:
"autômato finito M (eventualmente não determinístico e com transições em vazio)

38. Página 184:
Inserir, debaixo da linha "$\Sigma = \{a,b,c\}$" e antes de "Considere-se a cadeia $abbbc$":

$$\begin{aligned} P \;=\; & \{q_0 \to aq_1, \\ & q_1 \to bq_1, \\ & q_1 \to cq_2, \\ & q_1 \to q_2, \\ & q_2 \to cq_2, \\ & q_2 \to \varepsilon\} \end{aligned}$$

39. Página 221:
Onde se lê:
"uma sentença w de comprimento suficientemente longo pertencente à linguagem,"
Leia-se:
"uma sentença w de comprimento suficientemente longo pertencente à linguagem (sem, no entanto, especificar qual seria esta sentença),"

40. Página 222:
Substituir "Logo, L não é regular." por:
Portanto, $m*(1+\mid y \mid)$ é divisível por um número que não é 1 nem $m*(1+\mid y \mid)$. Logo. $m*(1+\mid y \mid)$ não é um número primo, a hipótese é falsa e L não é regular.

41. Página 235:
Inserir, antes do Teorema 3.34 e depois do Exemplo 3.85:
"Uma maneira alternativa de verificar se $L = \emptyset$ consiste em determinar o conjunto de estados acessíveis do autômato (Teorema 3.7) e verificar se ele contém algum estado final. Ou, ainda, determinar o conjunto de estados úteis do autômato (Teorema 3.8) e verificar se ele contém o estado inicial. Este método pode ser usado em qualquer tipo de autômato finito."

42. Página 272:
Acrescentar a seguinte nota de rodapé, depois de Backus-Naur Form[1].
[1] BNF é, muitas vezes, considerada como sigla de "Backus-Normal Form" ao invés de "Backus-Naur Form". Conforme observado por Knuth (citar Knuth64), no entanto, a segunda interpretação deve ser usada no lugar da primeira, por dois motivos: (i) em primeiro lugar, BNF não é, propriamente, uma Formal Normal, mas apenas uma Forma; (ii) em segundo lugar, a Forma foi criada por Backus e Naur e não apenas por Backus.

43. Página 341:
Inserir uma nova Seção 4.11, conforme abaixo:

4.11 Pertencimento

Dada uma gramática livre de contexto $G = (V, \Sigma, P, S)$ e uma cadeia $w \in \Sigma^*$, como determinar se $w \in Ł(G)$? A reposta para esta pergunta, fundamental na Teoria de Linguagens, é essencial para a construção de analisadores sintáticos, que por sua vez são necessários à construção de processadores de linguagens (compiladores e interpretadores).

Existem diversas maneiras de responder à esta pergunta, algumas simples e ineficientes, outras mais complexas porém eficientes. Iniciaremos com algoritmos de tempo exponencial, tendo por base as Formas Normais de Chomsky e de Greibach. Em seguida, apresentaremos o algoritmo CYK, que gera a resposta em tempo polinomial. Nas Seções 4.12 e 4.13 serão apresentadas técnicas de análise sintática determinística que permitem que a resposta seja gerada em tempo linear com o comprimento da cadeia de entrada (w).

Forma Normal de Chomsky

(a) Obter G' na Forma Normal de Chomsky ($A \rightarrow BC \mid a$), tal que $L(G) = L(G')$.

(b) Considerar $n = |w|$.

(c) Se $n > 0$, então fazer todas as derivações com $2*n-1$ passos.

(d) Se $n = 0$, então fazer todas as derivações com 1 passo.

(e) Se alguma dessas derivações gera w, então $w \in L(G)$.

(f) Caso contrário, $w \notin L(G)$..

A geração de uma cadeia de comprimento n numa gramática na Forma Normal de Chomsky demanda, necessariamente, a aplicação de $n-1$ regras do tipo $A \rightarrow BC$ (para gerar uma forma sentencial com n símbolos não-terminais) e também a aplicação de n regras do tipo $A \rightarrow a$ para transformar a forma sentencial numa sentença. Portanto, a geração de uma cadeia de n símbolos requer a aplicaçao de $2*n-1$ regras ou passos de derivação. Existe um conjunto finito de seqüências de derivação com qualquer quantidade de passos. Basta obter todas elas e verificar se alguma produz a cadeia informada na entrada. Em caso afirmativo, a cadeia pertence à linguagem. Em caso negativo, ela não pertence.

Suponha que o maior número de regras de um mesmo símbolo não terminal em G seja k. Então, uma árvore de derivação com $2n-1$ passos gera no máximo:

$$k^{2n-1}$$

formas sentenciais após os $2n-1$ passos. Em seguida, estas formas sentenciais devem ser inspecionadas para verificar se alguma delas corresponde à w e assim determinar a resposta do algoritmo. Logo, o tempo deste algoritmo é exponencial em n, onde $n = |w|$. Note que este tempo não leva em conta o tempo necessário para a geração de todas as formas sentenciais da árvore de derivação. O número total de formas sentenciais da árvore é:

$$1 + k + k^2 + ... + k^{2n-1} = \sum_{i=0}^{2n-1} k^i = \frac{k^{2n}-1}{k-1}$$

portanto o tempo total é exponencial também.

Exemplo 4.56 Considere G com as seguintes regras:

$$\begin{aligned} G &= (\{S,A,B,a,b,c\},\{a,b,c\},P,S) \\ P &= \{S \to AB \mid c, \\ &\quad A \to AA \mid a, \\ &\quad B \to BB \mid b\} \end{aligned}$$

Portanto, $k = 2$. Considere-se a cadeia abb:

- $n = |abb| = 3$.
- Deve-se pesquisar todas as seqüências de derivação com até $2 * 3 - 1 = 5$ passos.
- Para simplificar, serão consideradas apenas derivações mais à esquerda.

Na Figura 1 são apresentados os vários caminhos para a geração da cadeia aab em G. No máximo, 32 formas

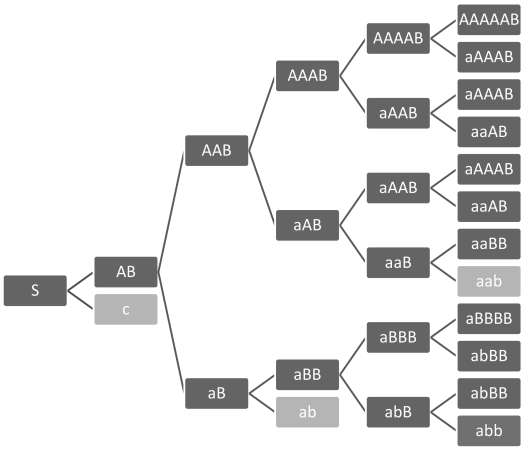

Figura 1: Sequências de derivações para a cadeia aab

sentenciais precisariam ser examinadas ($2^{2*3-1} = 2^5 = 32$). No entanto, para esta gramática em particular, apenas 12 formas sentenciais distintas são geradas.

Por último, cumpre notar que simular G diretamente pode não funcionar, pois pode haver seqüências infinitas de derivações em G. Por exemplo, se G possuir as regras unitárias $X \to Y$ e $Y \to X$, é possível que a seqüência de derivações torne-se infinita:

$$... \Rightarrow \alpha X \beta \Rightarrow \alpha Y \beta \Rightarrow \alpha X \beta \Rightarrow \alpha Y \beta \Rightarrow ...$$

Forma Normal de Greibach

Como todas as regras iniciam com um símbolo terminal, é suficiente gerar todas as formas sentenciais que produzem cadeias do comprimento da cadeia de entrada, e depois verificar se alguma delas corresponde à mesma.

(a) Obter G' na Forma Normal de Greibach, tal que $L(G') = L(G)$.

(b) Considerar $n = |w|$.

(c) Considerar k como o maior número de regras definido para um símbolo não terminal, entre todos os não terminais de G'.

(d) Obter todas as sequências de derivações mais à esquerda que geram formas sentenciais cujo prefixo seja uma cadeia de terminais de comprimento máximo n (existem no máximo k^n sequências distintas).

(e) Verificar se alguma dessas sequências de derivação corresponde à geração da cadeia w; em caso afirmativo, $w \in L(G)$; caso contrário, $w \notin L(G)$.

O número de formas sentenciais geradas por este método, com prefixo formado por n símbolos terminais, é:

$$k^n$$

Logo, o tempo é exponencial com o comprimento da cadeia de entrada n. Note que este tempo não leva em conta o tempo necessário para a geração de todas as formas sentenciais da árvore de derivação. O número total de formas sentenciais da árvore é:

$$1 + k + k^2 + ... + k^n = \sum_{i=0}^{n} k^i = \frac{k^{2n+1} - 1}{k - 1}$$

portanto o tempo total é exponencial também.

Exemplo 4.57 Considere-se a gramática a seguir, já apresentada na Forma Normal de Greibach:

$$\begin{aligned} G &= (\{S,B,C,a,b,c\},\{a,b,c\},P,S) \\ P &= \{S \to aBC \mid bBC, \\ &\quad B \to bB \mid b, \\ &\quad C \to c\} \end{aligned}$$

Considere-se a cadeia $abbc \in L(G)$. Então, $n = 4$ e $k = 2$ (pois existem duas produções para S, duas para B e apenas uma para C). Existem no máximo $2^4 = 16$ sequências distintas de derivações mais à esquerda que geram como prefixo uma cadeia de terminais de comprimento máximo 4, não havendo necessidade de se inspecionar outras sequências. A Tabela 1 relaciona todas as oito sequências (para o caso particular desta gramática) que geram cadeias de terminais de comprimento máximo 4. Observe que, no caso particular desta gramática, o número de formas sentenciais que deve ser inspecionado é apenas 6.

Tabela 1: Derivações mais à esquerda que geram prefixos de comprimento máximo 4

Sequência			ΣV^*		$\Sigma\Sigma V^*$		$\Sigma\Sigma\Sigma V^*$		$\Sigma\Sigma\Sigma\Sigma V^*$
1	S	$\Rightarrow$	aBC	$\Rightarrow$	$abBC$	$\Rightarrow$	$abbBC$	$\Rightarrow$	$abbbBC$
2								$\Rightarrow$	$abbbC$
3						$\Rightarrow$	$abbC$	$\Rightarrow$	$abbc$
4				$\Rightarrow$	abC	$\Rightarrow$	abc		
5		$\Rightarrow$	bBC	$\Rightarrow$	$bbBC$	$\Rightarrow$	$bbbBC$	$\Rightarrow$	$bbbbBC$
6								$\Rightarrow$	$bbbbC$
7						$\Rightarrow$	$bbbC$	$\Rightarrow$	$bbbc$
8				$\Rightarrow$	bbC	$\Rightarrow$	bbc		

Como se pode verificar, a terceira sequência de derivações produz a cadeia $abbc$. Por outro lado, a cadeia $bcbc$, também de comprimento 4, não pertence à linguagem, uma vez que nenhuma das oito sequências da Tabela 1 gera o prefixo $bcbc$ e, portanto, nenhuma sequência de derivações é capaz de gerar a cadeia $bcbc$. O tempo de execução deste algoritmo é proporcional a k^n, onde n é o comprimento da cadeia de entrada. Logo, esse tempo varia exponencialmente com o tamanho da cadeia, o que é um resultado considerado ineficiente e, portanto, indesejável do

ponto de vista prático.

Algoritmo CYK

O Algoritmo CYK[1], diferentemente dos algoritmos anteriores, de tempo exponencial, possui tempo polinomial, variando com o cubo do comprimento da cadeia de entrada ($O(n^3)$). Ao contrário daqueles, no entanto, o Algoritmo CYK apenas verifica se uma certa cadeia w pertence à linguagem gerada por uma gramática livre de contexto, não informando as regras que foram usadas na geração da mesma. Todos os algoritmos apresentados nesta seção funcionam para qualquer linguagem livre de contexto. O exemplo apresentado a seguir trata do pertencimento à uma linguagem livre de contexto não determinística. Os métodos discutidos nas Seções 4.12 e 4.13 possuem tempos lineares (portanto melhores do que o CYK), porém funcionam apenas para linguagens livres de contexto determinísticas.

O algoritmo funciona, em essência, identificando os símbolos não terminais de G que geram subcadeias w de comprimentos sucessivamente maiores, até que se chegue ao comprimento n. O Algoritmo CYK pressupõe que a gramática em questão, digamos $G = (V, \Sigma, P, S)$, está na Forma Normal de Chomsky. Ele considera a tabela apresentada a seguir:

n	X_{1n}						
$n-1$	$X_{1(n-1)}$	X_{2n}					
$n-2$	$X_{1(n-2)}$	$X_{2(n-1)}$	X_{3n}				
...							
3	X_{13}	X_{24}	X_{35}	...	$X_{(n-2)n}$		
2	X_{12}	X_{23}	X_{34}	...	...	$X_{(n-1)n}$	
1	X_{11}	X_{22}	X_{33}	...	...	...	X_{nn}
	σ_1	σ_2	σ_3	...	...	...	σ_n

Nesta tabela os símbolos $\sigma_1\sigma_2...\sigma_n$ dispostos na linha horizontal inferior representam a cadeia de entrada $w \in \Sigma^*$ com comprimento n. As linhas são numeradas de baixo para cima, de 1 até n, e referem-se aos comprimentos das diferentes subcadeias que podem ser encontradas em w. A tabela é então preenchida com X_{ij}, onde cada X_{ij} representa um conjunto de símbolos não terminais capazes de derivar a cadeia $\sigma_i\sigma_{i+1}...\sigma_j$. Por exemplo, na linha 2 encontramos X_{12}, X_{23}, X_{34}, ..., $X_{(n-1)n}$, cada um dos quais contém os símbolos não terminais capazes de derivar, respectivamente, $\sigma_1\sigma_2$, $\sigma_2\sigma_3$, $\sigma_3\sigma_4$, ..., $\sigma_{n-1}\sigma_n$.

Trata-se, portanto, de preencher a tabela com os valores de X_{ij}. Caso $S \in X_{1n}$, então $w \in L(G)$. Caso contrário, $w \notin L(G)$. O método para calcular X_{ij} é apresentado a seguir, e as linhas são preenchidas de baixo para cima (ou seja, da linha ! até a linha n).

No início, os valores de X_{ij}, com $i = j$, são determinados diretamente. Como se trata de cadeias de comprimento 1, e como G está na Forma Normal de Chomsky, a única possibilidade de geração destas cadeias unitárias é através de regras to tipo $A \to a$. Em outras palavras, X_{11} contém os símbolos não terminais capazes de derivar diretamente σ_1, X_{22} contém os símbolos não terminais capazes de derivar diretamente σ_2 e assim por diante. Portanto, a linha 1 da tabela é preenchida diretamente.

As demais linhas da tabela (2 até n, nesta ordem) são preenchidas como segue. Antes disso cumpre observar que, ao preecher uma linha, todas as linhas inferiores já estarão preenchidas. Desta maneira, todas as possibilidades de geração de cadeias de comprimento menor que o comprimento corrente (o que inclui prefixos e sufixos próprios), assim como os não terminais que as geram, já terão sido identificados.

Note que os índices i e j de X_{ij} não se referem, respectivamente, à linha e à coluna do mesmo, como seria usual. A linha onde X_{ij} ocorre é dada por $j - i + 1$ e a coluna corresponde ao valor de i.

Considere então X_{ij}, com $i \neq j$ (o caso $i = j$ foi tratado anteriormente e corresponde à linha 1 da tabela). O conjunto de símbolos não terminais que definem o valor de X_{ij} são os A tais que $A \Rightarrow \sigma_i...\sigma_j$. Como a geração não é direta,

[1] O nome refere-se às letras iniciais de Cocke, Younger e Kasami, respectivamente John Cocke, Daniel Younger e Tadao Kasami que, juntamente com Jacob T. Schwartz, descobriram o algoritmo de forma independente. O algoritmo foi publicado pela primeira vez por Itiroo Sakai em 1961 (citar sakai1961).

e G está na Forma Normal de Chomsky, segue que uma regra do tipo $A \rightarrow BC$ deve ser usada. Existem então $j-i$ possibilidades para se gerar esta cadeia:

- $B \Rightarrow \sigma_i$ e $C \Rightarrow \sigma_{i+1}...\sigma_j$.
- $B \Rightarrow \sigma_i\sigma_{i+1}$ e $C \Rightarrow \sigma_{i+2}...\sigma_j$.
- $B \Rightarrow \sigma_i\sigma_{i+1}\sigma_{i+2}$ e $C \Rightarrow \sigma_{i+3}...\sigma_j$.
- ...
- $B \Rightarrow \sigma_i\sigma_{i+1}\sigma_{i+2}...\sigma_{j-1}$ e $C \Rightarrow \sigma_j$.

ou seja, $B \Rightarrow \sigma_i...\sigma_k$ e $C \Rightarrow \sigma_{k+1}\sigma_j$ para $i \leq k < j$. Todas estas possibilidades devem então ser consideradas para determinar o valor de X_{ij}. Em outras palavras, para determinar se $A \in X_{ij}$ deve-se considerar todos os casos válidos em que:

- $i \leq k < j$.
- $B \in X_{ik}$.
- $C \in X_{(k+1)j}$.
- $A \rightarrow BC$ é uma regra de G.

. Note que, para cada X_{ij}, com $1 \leq i, j \leq n$, $i \neq j$, os seguintes pares são comparados:

- Começando com X_{ii} com $X_{(i+1)j}$.
- Depois $X_{i(i+1)}$ com $X_{(i+2)j}$.
- ...
- Até $X_{i(j-1)}$ com X_{jj}.

Portanto, o conjunto X_{ij} é obtido a partir de $X_{ii}X_{(i+1)j} \cup X_{i(i+1)}X_{(i+2)j} \cup ... \cup X_{i(j-1)}X_{jj}$, sendo formado pelo lado esquerdo das regras que possuem $X_{ii}X_{(i+1)j}$ ou $X_{i(i+1)}X_{(i+2)j}$ ou ... ou $X_{i(j-1)}X_{jj}$ do lado direito.

A Figura 2 ilustra esta situação. Observe que X_{ij} está na coluna i e por ele passa uma diagonal e uma vertical. As comparações são feitas entre os pares que estão na coluna (de baixo para cima, se aproximando de X_{ij}) com os pares que estão na diagonal (da esquerda para a direita, se afastando de X_{ij}). Por exemplo:

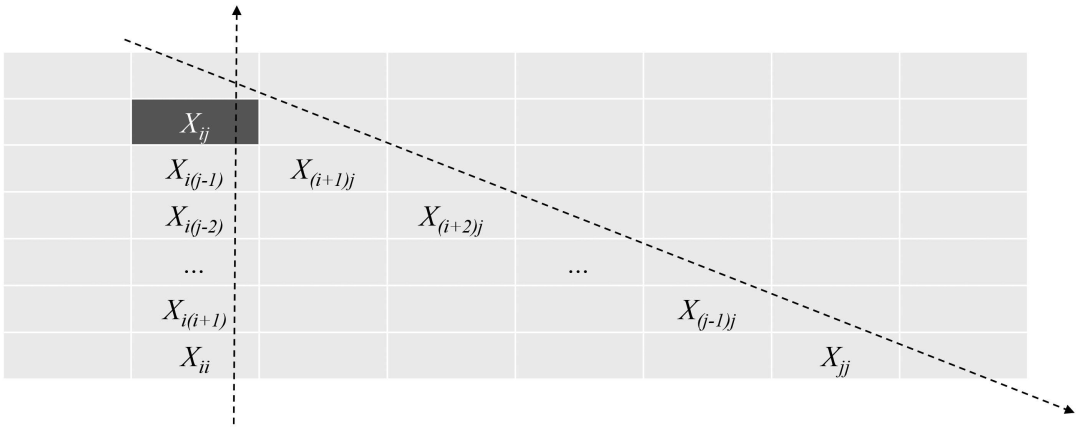

Figura 2: Cálculo de X_{ij}

- O cálculo de X_{12} envolve os pares X_{11} e X_{22} (para $k = 1$).
- O cálculo de X_{13} envolve os pares X_{11} e X_{23} (para $k = 1$) e depois os pares X_{12} e X_{33} (para $k = 2$).

- O cálculo de X_{14} envolve os pares X_{11} e X_{24} (para $k = 1$), depois os pares X_{12} e X_{34} (para $k = 2$) e finalmente os pares X_{13} e X_{44} (para $k = 3$).

Como a construção de cada X_{ij} demanda a comparação com até $n-1$ pares, segue que o tempo para calcular cada X_{ij} é $O(n)$, onde n é o comprimento da cadeia de entrada. Como a tabela possui:

$$\sum_{i=1}^{n} i = \frac{n*(n+1)}{2} = O(n^2)$$

elementos, então o tempo total do Algoritmo CYK é $O(n)*O(n^2) = O(n^3)$.

Exemplo 4.58 Considere a linguagem $\{ww^R \mid w \in \{a,b\}^+\}$ gerada pela gramática G abaixo:

$$\begin{aligned} G &= (\{S,a,b\},\{a,b\},P,S) \\ P &= \{S \to aSa \mid bSb \mid aa \mid bb\} \end{aligned}$$

Colocada na Forma Normal de Chomsky, G torna-se G' apresentada abaixo:

$$\begin{aligned} G' &= (\{S,A,B,X,Y,a,b\},\{a,b\},P,S) \\ P &= \{S \to AX \mid BY \mid AA \mid BB, \\ &\quad X \to SA, \\ &\quad Y \to SB, \\ &\quad A \to a, \\ &\quad B \to b\} \end{aligned}$$

Suponha agora que se deseja determinar se a cadeia $aa \in L(G)$.

2	$\{S\}$	
1	$\{A\}$	$\{A\}$
	a	a

Com $X_{11} = X_{22} = \{A\}$ e $X_{12} = \{S\}$ (pois $k = 1$, com X_{11} e X_{22}). Portanto, $aa \in L(G)$. Suponha agora que se deseja determinar se a cadeia $ab \in L(G)$.

2	$\{\}$	
1	$\{A\}$	$\{B\}$
	a	a

Com $X_{11} = \{A\}$, $X_{22} = \{B\}$ e $X_{12} = \{\}$ (pois para $k = 1$, não existe lado direito de regra para X_{11} e X_{22}). Portanto, $ab \notin L(G)$. Suponha agora que se deseja determinar se a cadeia $aabaabaa \in L(G)$.

8	$\{S\}$							
7	–	$\{X\}$						
6	–	$\{S\}$	–					
5	–	–	$\{X\}$	–				
4	–	–	$\{S\}$	–	–			
3	$\{Y\}$	–	–	$\{Y\}$	–	–		
2	$\{S\}$	–	–	$\{S\}$	–	–	$\{S\}$	
1	$\{A\}$	$\{A\}$	$\{B\}$	$\{A\}$	$\{A\}$	$\{B\}$	$\{A\}$	$\{A\}$
	a	a	b	a	a	b	a	a

onde:

- $X_{11} = X_{22} = X_{44} = X_{55} = X_{77} = X_{88} = \{A\}$ pois $A \to a$.
- $X_{33} = X_{66} = \{B\}$ pois $B \to b$.

- $X_{12} = \{S\}$ pois $k = 1$ e $S \to AA$.
- $X_{45} = \{S\}$ pois $k = 4$ e $S \to AA$.
- $X_{78} = \{S\}$ pois $k = 7$ e $S \to AA$.
- $X_{13} = \{Y\}$ pois $k = 2$ e $Y \to SB$.
- $X_{46} = \{Y\}$ pois $k = 5$ e $Y \to SB$.
- $X_{36} = \{S\}$ pois $k = 3$ e $S \to BY$.
- $X_{37} = \{X\}$ pois $k = 6$ e $X \to SA$.
- $X_{27} = \{S\}$ pois $k = 2$ e $S \to AX$.
- $X_{28} = \{X\}$ pois $k = 7$ e $X \to SA$.
- $X_{18} = \{S\}$ pois $k = 1$ e $S \to AX$.

Como $S \in X_{18}$, segue que $aabaabaa \in L(G)$. Note, este exemplo, que cada X_{ij} contém apenas um símbolo não terminal. No caso geral, X_{ij} pode conter vários símbolos não terminais, conforme eles tenham o mesmo lado direito satisfazendo as condições anteriores.

Análise determinística

Os algoritmos apresentados nesta seção determinam o pertencimento de uma cadeia a uma linguagem livre de contexto em tempo exponencial (usando a Forma Normal de Chomsky ou a Forma Normal de Greibach) ou cúbico com o comprimento da cadeia de entrada (CYK). Soluções ainda mais eficientes (de tempo linear), para classes restritas de gramáticas e linguagens livres de contexto, são discutidas nas Seções 4.12 e 4.13.

44. `Página 385:`
`Inserir uma nova Seção 4.15, conforme abaixo:`

4.15 Lema de Ogden

O *Pumping Lemma*, conforme explicado na Seção 4.14, se mostra válido não apenas para as linguagens livres de contexto, mas também para algumas linguagens que não são livres de contexto. Por isso, ele não pode ser usado para caracterizar as linguagens livres de contexto. A Figura 3 ilustra este fato.

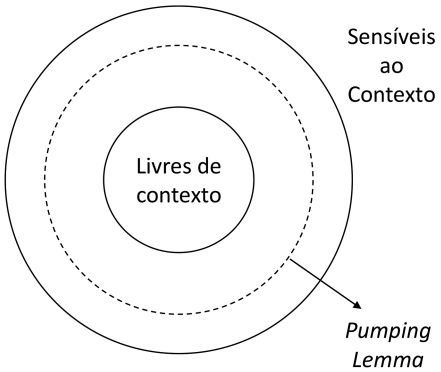

Figura 3: O *Pumping Lemma* e as linguagens livres de contexto

Exemplo 4.87 Considere a linguagem $L = \{a^i b^j c^k \mid i = j \vee j = k$ de forma exclusiva$\}$. Esta linguagem compreende, entre outras, as sentenças *abcc*, *aabbc*, *aabbccc* etc. Este linguagem, conforme será provado mais adiante, não é livre de contexto. No entanto, L satisfaz o *Pumping Lemma* e, portanto, o mesmo não pode ser usado para provar que L não é livre de contexto.

Suponha $z = uvwxy = a^n b^n c^m$, onde n é a constante do *Pumping Lemma* e $m \neq n$. Como $\mid z \mid = 2n + m$, esta sentença satisfaz os critérios para aplicação do *Pumping Lemma*. Supondo que L seja livre de contexto, o *Pumping Lemma* estabelece que $\mid vwx \mid \leq n$ e $\forall i, uv^i wx^i y \in L$.

Levando em conta este fato, temos que *vwx* pode ser formado apenas por símbolos *a*, ou por símbolos *a* seguidos de símbolos *b*, ou apenas por símbolos *b*, ou por símbolos *b* seguidos de símbolos *c* ou, ainda, apenas símbolos *c*. Se considerarmos este último caso, então *vx* contém apenas por símbolos *c* (um ou mais). Ao bombearmos *v* e *x* não há garantias de que a cadeia resultante não pertence a *L* (pois, para isto acontecer, a quantidade de *b* deveria ser igual à quantidade de *c*). Portanto não é possível mostrar que nesta quebra que cadeia resultante, após o bombeamento, não pertence a linguagem.

Lembre-se: para provar que *L* não é livre de contexto, é necessário considerar todas as possíveis quebras *uvwxy* e mostrar que, para todas elas, gera-se cadeias que não pertencem a *L*. Para provar que *L* não é livre de contexto, será necessário recorrer ao Lema de Ogden.

A Figura 4 ilustra o fato de que *L* não é livre de contexto mas ainda assim o *Pumping Lemma* falha na prova.

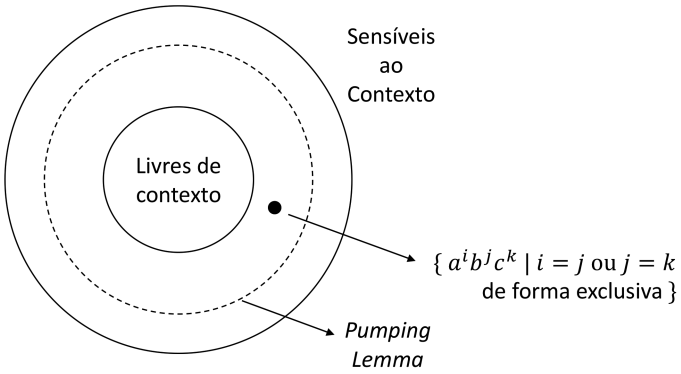

Figura 4: Linguagem que não pode ser provada não ser livre de contexto pelo *Pumping Lemma*

⋆

O Lema de Ogden é uma versão mais forte do *Pumping Lemma*. Ele é válido para todas as linguagens livres de contexto, conforme provado a seguir, mas, assim como o *Pumping Lemma*, também é válido para algumas linguagens que não são livres de contexto. A sua vantagem, no entanto, é que o conjunto de linguagens que não são livres de contexto e para o qual o Lema de Ogden se mostra válido é menor do que o conjunto das linguagens que não são livres de contexto e para o qual o *Pumping Lemma* é válido. Este fato é ilustrado na Figura 5. Em citar wise1976, David Wise fornece uma condição necessária e suficiente para caracterizar uma linguagem como sendo livre de contexto. Este texto é baseado em citar hopcroft1979 e citar shallit2008.

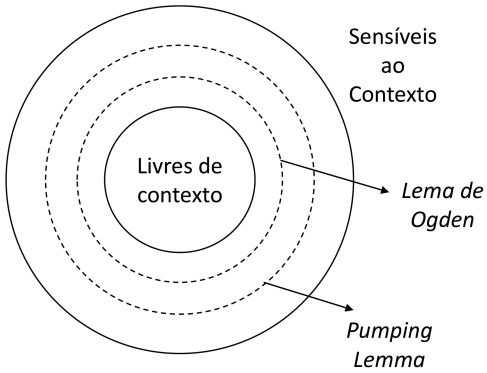

Figura 5: *Pumping Lemma* × Lema de Ogden

O Lema de Ogden pode ser usado, portanto, para mostrar que uma linguagem não é livre de contexto quando o *Pumping Lemma* falhar. Nem sempre, no entanto, o Lemma de Ogden poderá provar que uma linguagem não é livre de contexto. Formulado por William F. Odgen em 1968 (citar ogden1968), o seu objetivo inicial era provar que certas linguagens são inerentemente ambíguas.

Teorema 4.23 (Lemma de Ogden) Seja L uma linguagem livre de contexto. Então, existe uma constante n tal que, para toda sentença $z \in L$, $\mid z \mid \geq n$, em que forem marcados[2] n ou mais símbolos, então $z = uvwxy$ e (i) vx tem pelo menos um símbolo marcado, (ii) vwx tem no máximo n símbolos marcados e (iii) $\forall i \geq 0, uv^i wx^i y \in L$.

Para provar, consideramos $n = 2^{k+1}$, onde k é o número de símbolos não-terminais da gramática G na Forma Normal de Chomsky que gera L. Da mesma forma que no *Pumping Lemma*, deve-se construir um caminho C que vá da raiz da árvore de sintaxe que representa z até alguma folha (representada por um símbolo terminal).

O caminho C deverá ser construído colocando-se a raiz da árvore como o vértice inicial de C. Em seguida (lembre-se que a gramática está na Forma Normal de Chomsky):

(a) Se o último vértice de C é um símbolo terminal, o processo de formação de C termina.

(b) Se o último vértice de C é um símbolo não-terminal com um único filho, escolher este como próximo vértice de C.

(c) Se o último vértice de C é um símbolo não-terminal com dois filhos:

   i. Se ambos os filhos geram subárvores com símbolos marcados, escolher como próximo vértice de C o símbolo não-terminal que possui a maior quantidade de símbolos marcados; se as quantidades forem idênticas, escolher arbitrariamente o próximo vértice de C. Em qualquer caso, o pai deste dois filhos é denominado "ponto de ramificação" (PR, do inglês *branch point*).

   ii. Se um filho gerar uma subárvore com símbolos marcados e o outro não, escolher o primeiro como próximo vértice de C.

Desta forma, o caminho C é formado por símbolos não-terminais terminando sempre com um símbolo terminal. Entre os símbolos não-terminais de C existirão alguns que são pontos de ramificação e outros que não o são.

Considere agora apenas os pontos de ramificação de C (que comparecem em vértices não necessariamente successivos). Pela maneira como C é construído, é fácil notar que, se PR_i e PR_{i+1} são pontos de ramificação consecutivos em C, isto é, PR_{i+1} aparece logo depois de PR_i em C, então o número de símbolos marcados na subárvore gerada por PR_{i+1} é no mínimo a metade dos símbolos marcados na subárvore gerada por PR_i. Logo, devem existir pelo menos $k+1$ pontos de ramificação em C. Senão vejamos. Suponha que o caminho C possui um único ponto de ramificação:

$$C = ... \rightarrow PR_1 \rightarrow ... \rightarrow T$$

onde PR_1 é um ponto de ramificação e T é um símbolo terminal. Neste caso, PR_1 gera dois símbolos marcados ($2^1 = 2$). De fato, se PR_1 é um ponto de ramificação, é porque se trata de um símbolo não-terminal com dois filhos, sendo que cada filho gera uma subárvore com um único símbolo. Suponha agora que C possui dois pontos de ramificação:

$$C = ... \rightarrow PR_2 \rightarrow ... \rightarrow PR_1 \rightarrow ... \rightarrow T$$

Neste caso, PR_2 gera uma cadeia com no máximo 4 (2^2) símbolos marcados (pois PR_1 gera uma subárvore com pelo menos a metade dos símbolos marcados de PR_2). Suponha agora que C possui k pontos de ramificação:

$$C = ... \rightarrow PR_k \rightarrow ... \rightarrow PR_2 \rightarrow ... \rightarrow PR_1 \rightarrow ... \rightarrow T$$

Neste caso, PR_k gera uma cadeia com no máximo 2^k símbolos marcados. O mesmo, portanto, vale para a raiz da árvore. No entanto, como é sabido que a árvore representa uma cadeia com um número mínimo

[2] A marcação de sum símbolo pode ser entendida como qualquer representação gráfica – como um sublinhado, um negrito ou uma cor diferente — que destaque o símbolo na cadeia.

de 2^{k+1} símbolos marcados (n), segue que C deve conter pelo menos $k+1$ pontos de ramificação. Com $k+1$ pontos de ramificação consegue-se um máximo de 2^{k+1} símbolos marcados, que é a quantidade mínima de símbolos marcados em z. Logo, devem existir no mínimo $k+1$ pontos de ramificação em C.

Sabendo que C contém pelo menos $k+1$ pontos de ramificação, considere agora os $k+1$ pontos de ramificação presentes no final de C (ou seja, mais distantes da raiz e mais próximos da folha da árvore). Como G possui apenas k símbolos não terminais, é fato que pelo menos um deles deve ser repetido neste trecho de C. Logo, neste trecho C possui dois pontos de ramificação associados ao mesmo símbolo não terminal. Digamos que este símbolo seja X. Veja a Figura 6.

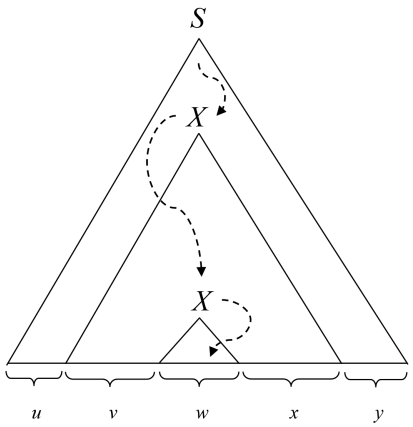

Figura 6: Caminho C

O primeiro ponto de ramificação pode, portanto, ser o de ordem $k+1$ e com isso vwx pode conter no máximo $2^{k+1} = n$ posições marcadas. Para calcular a quantidade mínima de símbolos marcados em vx, basta considerar que o X anterior possui dois filhos e é um ponto de ramificação. A cadeia w, por outro lado, é gerada por um destes dois filhos. Logo, v ou x contém pelo menos um símbolo marcado.

A partir deste ponto, a prova procede como no caso do *Pumping Lemma*:

$$\begin{aligned} S &\Rightarrow^* uXy \\ X &\Rightarrow^* vXx \\ X &\Rightarrow^* w \end{aligned}$$

ou seja, $X \Rightarrow uv^i wx^i y, \forall i \geq 0$. □

Exemplo 4.88 Suponha que o último vértice de um caminho qualquer C seja tal o número de símbolos marcados à esquerda seja zero e à direita diferente de zero. Neste caso, este vértice é adicionado ao caminho C, sem no entanto caracterizar um ponto de ramificação. Veja a Figura 7.

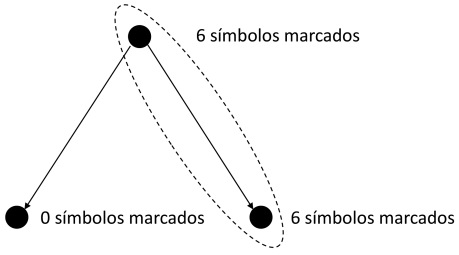

Figura 7: Vértice de C no qual um dos filhos não possui nenhum símbolo marcado

Suponha agora que o último vértice de um caminho qualquer C seja tal o número de símbolos marcados à esquerda seja diferente de zero e à direita também diferente de zero, porém um diferente do outro. Neste caso, o vértice que é adicionado ao caminho C é aquele que possui a maior quantidade de símbolos marcados e o novo vértice é considerado um ponto de ramificação. Veja a Figura 8.

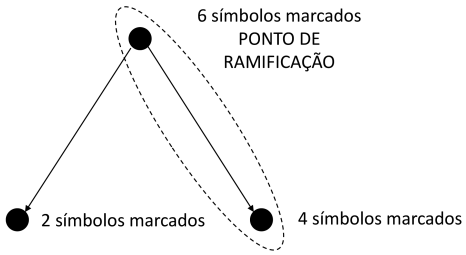

Figura 8: Ponto de ramificação de C no qual os filhos possuem quantidades diferentes de símbolos marcados

Finalmente, suponha que o último vértice de um caminho qualquer C seja tal o número de símbolos marcados à esquerda seja diferente de zero e à direita também diferente de zero, porém ambos iguais. Neste caso, o vértice que é adicionado ao caminho C pode ser qualquer um dos dois e o novo vértice é considerado um ponto de ramificação. Veja a Figura 9.

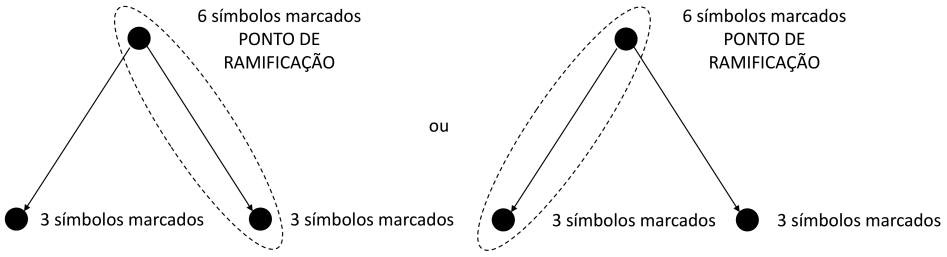

Figura 9: Ponto de ramificação de C no qual os filhos possuem quantidades idênticas de símbolos marcados

A linha tracejada representa, em todos os casos, vértices de C (que podem ou não ser pontos de ramificação). ⋆

O *Pumping Lemma* requer uma cadeia de comprimento mínimo $2^{k-1}+1$ a fim de que a árvore de derivação possua um caminho com pelo menos $k+1$ simbolos não terminais, o que garante a existência de pelo menos um símbolo não terminal repetido neste caminho, o que por sua vez permite o bombeamento das cadeias v e x.

O Lema de Ogden, por sua vez, requer uma cadeia de comprimento mínimo 2^{k+1} a fim de que a árvore de derivação possua um caminho com pelo menos $k+1$ pontos de ramificação, o que por sua vez garante a exsitência de pelo menos um símbolo não terminal repetido neste caminho, e finalmente o bombeamento das cadeias v e x.

A diferença entre o Lema de Ogden e o *Pumping Lemma* é que o primeiro bombeia apenas cadeias com símbolos marcados, independentemente do seu comprimento, ao passo que o segundo bombeia cadeias de comprimentos definidos. Note que se todos os símbolos da cadeia de entrada z forem marcados, então o enunciado do Lema de Ogden se torna o enunciado do *Pumping Lemma* (se vx possui pelo menos um símbolo marcado, e não existem símbolos não marcados, então $\mid vx \mid \geq 1$; além disso se vwx possui no máximo n símbolos marcados, e não existem símbolos não marcados, então $\mid vwx \mid \leq n$). Assim, o *Pumping Lemma* é um caso particular do Lema de Ogden.

Exemplo 4.88 Consideremos novamente a linguagem $L = \{a^i b^j c^k \mid i = j \vee j = k$ de forma exclusiva$\}$. Mostraremos, a seguir, que esta linguagem não é livre de contexto pela aplicação do Lema de Odgen (lembre-se de que não foi possível provar isto por meio do *Pumping Lemma*). A Figura 10 ilustra este fato.

Seja $z = uvwxy = a^n b^n c^{n+n!}$, onde n é a constante do Lema de Ogden, e suponha que todos os símbolos a estão marcados. Portanto, $z \in L$ e $\mid z \mid = 3n+n! \geq n$. Examinemos agora a cadeia vx. De acordo com o Lema de Ogden, vx deve conter pelo menos um símbolo marcado, ou seja, vx deve conter pelo menos um símbolo a (uma vez que todos os símbolos a foram marcados). Existem duas possibilidades para v e x, levando em contato que os símbolos marcados são um prefixo de z:

- $v = \varepsilon$ e x contém um símbolo a, ou

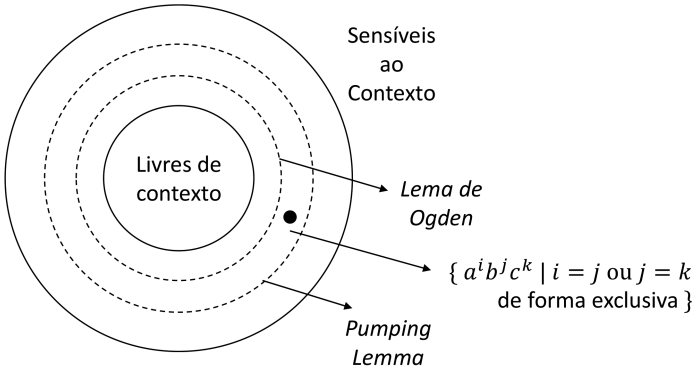

Figura 10: Linguagem que é provada não ser livre de contexto pelo Lema de Ogden

- v contém um símbolo a e x é qualquer.

A seguir, analisamos as duas possibilidades anteriores, na tentativa de determinar as cadeias v e x:

- Se $v = \varepsilon$ e x contém um símbolo a, então x pode conter:
  - Apenas símbolos a, ou
  - Símbolos a seguidos de símbolos b ou
  - Símbolos a seguidos de símbolos b e símbolos c.

  Nos dois últimos casos a cadeia uv^2wx^2y não pertence a L pois os símbolos estarão fora de ordem. No primeiro caso, podemos escrever v como a^m, com $m \geq 1$. Portanto, $uv^2wx^2y = a^{n+m}b^nc^{n+n!} \notin L$ pois $i \neq j$ e $j \neq k$.
- Se v contém um símbolo a e x é qualquer, então v pode conter:
  - Apenas símbolos a, ou
  - Símbolos a seguidos de símbolos b, ou
  - Símbolos a seguidos de símbolos b e símbolos c.

  Nos dois últimos casos a cadeia uv^2wx^2y não pertence a L pois os símbolos estarão fora de ordem. No primeiro caso, podemos escrever v como a^p, com $p \geq 1$. Por sua vez, x pode conter:
  - Apenas símbolos a, ou
  - Apenas símbolos b, ou
  - Apenas símbolos c, ou
  - Símbolos a seguidos de símbolos b, ou
  - Símbolos b seguidos de símbolos c, ou
  - Símbolos a seguidos de símbolos b e símbolos c.

  Nos três últimos casos a cadeia uv^2wx^2y não pertence a L pois os símbolos estarão fora de ordem. Portanto, $x = a^q$ ou $x = b^q$ ou $x = c^q$ para algum q. Se $x = a^q$ então $uv^2wx^2y \notin L$ pois $i \neq j$ e $j \neq k$. Se $x = c^q$ então $uv^2wx^2y \notin L$ pois $i \neq j$ e $j \neq k$. Resta analisar o caso $x = b^q$, conforme dos valores de p e q:
  - Se $p \neq q$, então basta escolher $i = 0$. Neste caso, temos que $i \neq j$ (pois as quantidades de a e b que eram originalmente iguais, ao subtrair quantidades diferentes tornam-se diferentes) e $j \neq k$ (pois j torna-se ainda menor do que k), portanto $uwy \notin L$.
  - Se $p = q$, então basta escolher $i = \frac{n!}{p} + 1$. Neste caso, temos que $uv^{\frac{n!}{p}+1}wx^{\frac{n!}{p}+1}y$ deveria pertencer a L. Substituindo v por a^p e x por b^p obtemos $ua^pa^{n!}wb^pb^{n!}c^{n+n!}$ ou ainda $uva^{n!}wxb^{n!}c^{n+n!}$. Como $uvwxy = a^nb^nc^{n+n!}$, segue que $uva^{n!}wxb^{n!}c^{n+n!} = ua^{n+n!}wb^{n+n!}c^{n+n!}$, ou seja $i = j = k$ e portanto a cadeia resultante não pertence a L.

Em todos os casos, mostramos que todas as quebras possíveis de z geram pelo menos uma cadeia que não pertence a L. Logo, L não é livre de contexto. ⋆

Exemplo 4.89 Considere agora a linguagem $L = \{a^ib^jc^k \mid i \neq j \wedge j \neq k \wedge i \neq k\}$. Mostraremos, a seguir, que esta linguagem não é livre de contexto pela aplicação do Lema de Odgen.

Considere a cadeia $z = a^n b^{n+n!} c^{n+2n!}$, onde n é a constante do Lema de Ogden, com todos os símbolos a marcados. Claramente, $z \in L$ e $| z | \geq n$. Então, de acordo com o Lema de Ogden, $z = uvwxy$ com as respectivas propriedades. Assim como no exemplo anterior, vamos tentar inferir a respeito das cadeias v e x.

Se v ou x contiverem dois símbolos distintos, então $uv^2wx^2y \notin L$ (pois isso altera a ordem dos símbolos). Portanto, v e x são formadas por um único símbolo. De acordo com o Lema de Ogden, v ou x devem conter pelo menos um símbolo a (pois apenas os símbolos a são marcados). Portanto:

- $v \in a^+$ e $x \in b^*$, ou
- $v \in a^+$ e $x \in c^*$, ou
- $v \in a^*$ e $x \in a^+$

Consideremos os três casos acima.

No primeiro caso ($v \in a^+$ e $x \in b^*$), seja $p = | v |$. Portanto, $v = a^p$. Como $v \in a^+$ e existem n símbolos a marcados, segue que $1 \leq p \leq n$. Como consequência, segue que p divide $n!$. Seja $q = \frac{n!}{p}$ e considere $i = 2q+1$. Logo:

$$uv^{2q+1}wx^{2q+1}y \in L$$

Substituindo v por a^p obtemos:

$$ua^{2pq+p}wx^{2q+1}y$$

Substituindo pq por $n!$ obtemos:

$$ua^{2n!+p}wx^{2q+1}y$$

Considere agora $i = 0$. Como $v = a^p$, então uwy contém $n-p$ símbolos a. Então, a cadeia acima (com $i = 2q+1$) contém $(n-p)+(2q+1)p = n-p+2pq+p = n+2n!$ símbolos a. Como nem v nem x contém símbolos c, a quantidade deste símbolo permanece inalterada, ou seja, $n+2n!$. Consequentemente $i = k$ e a cadeia resultante não pertence a L.

No segundo caso ($v \in a^+$ e $x \in c^*$), seja $p = | v |$. Portanto, $v = a^p$. Como $v \in a^+$ e existem n símbolos a marcados, segue que $1 \leq p \leq n$. Como consequência, segue que p divide $n!$. Seja $q = \frac{n!}{p}$ e considere $i = q+1$. Logo:

$$uv^{q+1}wx^{q+1}y \in L$$

Substituindo v por a^p obtemos:

$$ua^{pq+p}wx^{q+1}y$$

Substituindo pq por $n!$ obtemos:

$$ua^{n!+p}wx^{q+1}y$$

Considere agora $i = 0$. Como $v = a^p$, então uwy contém $n-p$ símbolos a. Então, a cadeia acima (com $i = q+1$) contém $(n-p)+(q+1)p = n-p+pq+p = n+n!$ símbolos a. Como nem v nem x contém símbolos b, a quantidade deste símbolo permanece inalterada, ou seja, $n+n!$. Consequentemente $i = j$ e a cadeia resultante não pertence a L.

No terceiro caso ($v \in a^*$ e $x \in a^+$), seja $p = | v |$ e $q = | x |$. Portanto, $v = a^p$ e $x = a^q$. Considere $i = \frac{n!}{p+q} + 1$. Logo:

$$uv^{(\frac{n!}{p+q}+1)}wx^{(\frac{n!}{p+q}+1)}y \in L$$

Substituindo v por a^p e x por a^q obtemos:

$$ua^{p\cdot(\frac{n!}{p+q}+1)}wa^{q\cdot(\frac{n!}{p+q}+1)}y$$

Considere agora $i = 0$. Como $v = a^p$ e $x = a^q$, então uwy contém $n-p-q$ símbolos a. Então, a cadeia acima (com $i = \frac{n!}{p+q}+1$) contém $(n-p-q)+(\frac{n!}{p+q}+1)\cdot p+(\frac{n!}{p+q}+1)\cdot q = n+n!$ símbolos a. Como nem v nem x contém símbolos b, a quantidade deste símbolo permanece inalterada, ou seja, $n+n!$. Consequentemente $i = j$ e a cadeia resultante não pertence a L.

Em todos os casos, mostramos que todas as quebras possíveis de z geram pelo menos uma cadeia que não pertence a L. Logo, L não é livre de contexto. ⋆

Como mostram os exemplos anteriores, a aplicação do Lema de Ogden, para provar que uma linguagem não é livre de contexto, é mais elaborada do que a aplicação do *Pumping Lemma*. Deve-se não apenas escolher uma sentença adequada, como também escolher os símbolos marcados e usar de bastante engenhosidade para mostrar que o Lema de Ogden não é observado. Por esses motivos, o uso do Lema de Ogden é normalmente restrito aos casos em que o *Pumping Lemma* não é capaz de provar que a linguagem em questão não é livre de contexto.

45. `Página 392:`
    `Remover todo o conteúdo a partir de` “Conforme o Algoritmo 4.15” `até` “presente publicação.” `(página 394).`
    `Adicionar no lugar` “Conforme a Seção 4.11”

46. `Página 394:`
    `Substituir` “Por outro lado, conforme o Algoritmo 4.11” `por` “Conforme o Algoritmo 4.11”

47. `Página 395:`
    `Inserir, ao final do parágrafo` “Uma maneira alternativa...”
    “Em outras palavras, basta verificar se a raiz da gramática é um símbolo útil (Teorema 4.5). Este método pode ser usado em qualquer tipo de gramática livre de contexto.”

48. `Página 444:`
    `Inserir uma nova Seção 5.3, conforme abaixo:`

5.3 Gramáticas de atributos

Concebidas originalmente por Knuth (em citar knuth1968), com extensões fornecidas por Wegner (citar knuth1990), as gramáticas de atributos são a culminação de um trabalho iniciado por Edgar T. Irons com as linguagens Algol 60 e IMP, também na década de 1960 (citar irons1961). A história da invenção das gramáticas de atributos é contada em detalhes em citar knuth1990. Um pequeno resumo com exemplos pode ser encontrado em citar sebesta2012. Textos mais completos e detalhados estão disponíveis em citar fischer1988 (Capítulo 14), citar slonneger1995 (Capítulo 3) e citar waite1984 (Capítulo 8).

Uma gramática de atributos é uma gramática livre de contexto estendida com atributos, funções de avaliação dos atributos e, eventualmente, predicados (ou condições). Gramáticas de atributos se prestam para dois objetivos principais (os quais, como o leitor poderá perceber, estendem o poder das gramáticas livres de contexto subjacentes):

- Elas podem ser usadas para representar as dependências de contexto de uma linguagem de programação (por exemplo, verificando se os identificadores foram declarados, se os tipos dos operadores são compatíveis entre si etc). Alguns autores chamam, erronamente, tais dependências de contexto de semântica estática.
- Elas podem ser usadas para representar a semântica (ou seja, o significado) de uma sentença calculando, por exemplo, o valor númerico de uma expressão aritmética ou gerando um código para uma outra máquina que representa a semântica do comando traduzido.

No primeiro caso, uma gramática de atributos pode substituir com vantagens uma gramática sensível ao contexto, normalmente difícil de ser lida e construída. No segundo caso, é possível especificar um compilador inteiro por meio de uma gramática de atributos, e usar a mesma para gerar automáticamente tal compilador (principalmente para verificação de contexto e geração de código). Tipicamente gramáticas de atributos servem aos dois propósitos, sendo usadas, direta ou indiretamente, na construção de compiladores.

Gramáticas de atributos foram bastante usadas e objeto de muita pesquisa no final da década de 1960 e na década de 1970. Depois disso elas caíram em desuso. Apesar de serem relativamente simples, gramáticas de atributos para linguagens de programação reais podem se tornar muito grandes e complexas, podendo inclusive impactar de forma significativa o desempenho dos compiladores construídos a partir das mesmas.

Ainda assim, a ideia fundamental das gramáticas de atributos (ou seja, o enriquecimento de uma gramática livre de contexto com atributos) permanece atual, estando presente em virtualmente todos os compiladores existentes, e dando origem a uma técnica importante de compilação, conhecida como Tradução Dirigida por Sintaxe (do inglês *SDT — Syntax Directed Translation*), em que o analisador sintático comanda toda a operação do compilador (citar aho2006). Neste contexto, as gramáticas de atributos recebem o nome de Definições Dirigidas por Sintaxe (do inglês *SDD — Syntax Directed Definition*).

Algumas ferramentas, como é o caso do YACC (citar johnson1978, sigla de “*yet-another-compiler-compiler*”) aceitam como entrada uma gramática de atributos na qual a gramática livre de contexto é enriquecida com funções de avaliação dos atributos escritos na linguagem C. Como resultado, o YACC é capaz de gerar um compilador

completo a partir da gramática de atributos fornecida na entrada. O analisador sintático ascendente gerado pelo YACC é complementado pelas funções especificadas pelo usuário, nos pontos em que elas devem ser acionadas. Em um caso como o do YACC, a gramática de atributos da entrada é capaz de efetuar tanto a verificação das dependências de contexto da linguagem-fonte, quanto as ações semânticas necessárias à geração do código-objeto.

Seja $G = (V, \Sigma, P, S)$ uma gramática livre de contexto. Uma gramática de atributos é construída a partir de G, da seguinte foma:

- Cada símbolo (terminal ou não terminal) de G está associado a uma quantidade finita de atributos (ou variáveis, no sentido usual do termo). Assim, se $X \in V$, então $A(X)$ denota o conjunto de atributos de X.
- Os atributos de um símbolo podem ser de dois tipos: sintetizados (S) ou herdados (H). Portanto, $A(X) = S(X) \cup H(X)$.
- Cada regra da gramática (elemento de P) está associada a zero ou mais funções de avaliação de atributos. Uma função de avaliação de atributos é um comando de atribuição com um atributo (variável) do lado esquerdo e uma expressão (formada por variáveis e literais) do lado direito. Se $U \rightarrow XY...Z \in P$, com $U \in N$ e $X, Y, ..., Z \in V$, então deve haver uma função para cada atributo sintetizado de U e uma função para cada atributo herdado de X, de Y, ... e de Z. Ou seja, supondo que U possua us atributos sintetizados e uh atributos herdados, que X possua xs atributos sintetizados e xh atributos herdados e assim por diante:
  - $A(U) = S(U) \cup H(U)$, com $S(U) = \{su_1, ..., su_{us}\}$ e $H(U) = \{hu_1, ..., hu_{uh}\}$
  - $A(X) = S(X) \cup H(X)$, com $S(X) = \{sx_1, ..., sx_{xs}\}$ e $H(X) = \{hx_1, ..., hx_{xh}\}$
  - $A(Y) = S(Y) \cup H(Y)$, com $S(Y) = \{sy_1, ..., sy_{ys}\}$ e $H(Y) = \{hy_1, ..., hy_{yh}\}$
  - ...
  - $A(Z) = S(Z) \cup H(Z)$, com $S(Z) = \{sz_1, ..., sz_{zs}\}$ e $H(Z) = \{hz_1, ..., hz_{zh}\}$

  Então, o seguinte conjunto de funções de avaliação de atributos devem ser definidas para a regra $U \rightarrow XY...Z$ de G:
  - su_1 é função dos argumentos contidos em $H(U) \cup A(X) \cup A(Y) \cup ... \cup A(Z)$
  - ...
  - su_{us} é função dos argumentos contidos em $H(U) \cup A(X) \cup A(Y) \cup ... \cup A(Z)$
  - hx_1 é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$
  - ...
  - hx_{xh} é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$
  - hy_1 é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$
  - ...
  - hy_{yh} é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$
  - ...
  - hz_1 é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$
  - ...
  - hz_{zh} é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$

  Note, acima, que são apresentadas uma função de avaliação de atributos para cada atributo sintetizado de U e uma função de avaliação de atributos para cada atributo herdado de $X, Y, ..., Z$.

  Note, também, que o valor de um atributo sintetizado correspondente ao lado esquerdo de uma regra pode ser função dos tributos herdados do mesmo símbolo e dos atributos (quaisquer) dos símbolos que comparecem do lado direito da regra. Por outro lado, o valor de um atributo herdado que comparece no lado direito de uma regra pode ser função dos atributos (quaisquer) do símbolo do lado esquerdo da regra e dos atributos sintetizados dos símbolos que comparecem do lado direito da regra. A definição dos atributos que podem ser usados para produzir o valor de um outro atributo varia de autor para autor. A convenção adotada aqui é de citar slonneger1995.

  No caso particular de U possuir apenas um atributo sintetizado e nenhum herdado, e cada um dos símbolos X, Y, ... e Z possuir apenas um atributo herdado e nenhum sintetizado, então as funções de avaliação de atributos para esta regra se reduzem à:

  - *su* é função dos argumentos contidos em $H(U) \cup A(X) \cup A(Y) \cup ... \cup A(Z)$
  - *hx* é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$
  - *hy* é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$
  - ...
  - *hz* é função dos argumentos contidos em $A(U) \cup S(X) \cup S(Y) \cup ... \cup S(Z)$

- Cada regra de G está associada a um conjunto (eventualmente vazio) de predicados ou condições. Um predicado é uma proposição construída a partir dos atributos dos símbolos da regra em questão, e deve produzir VERDADEIRO após a aplicação da regra gramatical e avaliação dos seus atributos. Se o predicado produzir resultado FALSO a cadeia de entrada é rejeitada. Considerando novamente a regra $U \rightarrow XY...Z$, um predicado pode ser expresso como uma proposição usando como argumentos os elementos de $A(U) \cup A(X) \cup A(Y) \cup ... \cup A(Z)$.

  A definição original das gramáticas de atributos não contempla os predicados. Estes são introduzidos apenas para facilitar a interrupção da análise caso alguma regra de contexto seja violada pela cadeia de entrada. De qualquer forma, a condição de parada com rejeição pode ser simulada pelas funções de avaliação de atributos com a chamada de alguma rotina que execute esta função. O uso de predicados associados às regras gramaticais é mencionado em citar sebesta2012; eles também são usados em citar waite1984, slonneger1995, mas com o nome de "condições".

Observações importantes:

(a) Cada função de avaliação de atributos pode referenciar apenas os atributos da regra em questão.

(b) Atributos sintetizados dos símbolos $X, Y, ...Z$ e atributos herdados de U são avaliados pelas respectivas funções, quando estes símbolos comparecerem na árvore respectivamente como pais e filhos.

(c) Deve-se evitar circularidade (por exemplo $x = y$ e $y = x$) pois nestes casos não é possível avaliar os atributos.

(d) Os atributos herdados da raiz da gramática (S) são argumentos recebidos do sistema e os atributos sintetizados dos símbolos terminais (os *tokens* da linguagem) são os respectivos valores.

(e) Quado o mesmo símbolo comparecer mais de uma vez numa mesma regra, serão usados índices para diferenciá-los conforme a posição em que ocorrem na regra.

(f) Funções de avaliação de atributos são envolvidas por linhas finas, como em exemplo.

(g) Predicados são envolvidos por linhas grossas, como em exemplo.

Em linhas gerais, atributos sintetizados servem para transferir dados de baixo para cima na árvore sintática. Atributos herdados servem para transferir dados de cima para baixo. Desta maneira, dados relevantes podem circular livremente pela árvore de sintaxe, tendo os seus valores definidos (ou atualizados) e consultados nos pontos onde houver necessidade.

Exemplo 1:
Conside a gramática do Exemplo 4.7, porém modificadade tal forma que, no lugar de a e b, sejam usados os algarismos 0 até 9:

$$
\begin{aligned}
E &\rightarrow T + E \\
E &\rightarrow T \\
T &\rightarrow F * T \\
T &\rightarrow F \\
F &\rightarrow 0 \\
F &\rightarrow 1 \\
F &\rightarrow 2 \\
F &\rightarrow 3
\end{aligned}
$$

$$
\begin{aligned}
F &\rightarrow 4 \\
F &\rightarrow 5 \\
F &\rightarrow 6 \\
F &\rightarrow 7 \\
F &\rightarrow 8 \\
F &\rightarrow 9 \\
F &\rightarrow (E)
\end{aligned}
$$

Esta gramática gera sentenças como $1+2$, $3+4*5$ e $(3+4)*(5+6)$. A gramática de atributos apresentada a seguir transforma estas expressões no valor numérico das mesmas, conforme a ordem usual de avaliação:

$E \rightarrow T + E$
E_1.val := T.val + E_2.val

$E \rightarrow T$
E.val := T.val

$T \rightarrow F * T$
T_1.val := F.val * T_2.val

$T \rightarrow F$
T.val := F.val

$F \rightarrow 0$
F.val := 0

$F \rightarrow 1$
F.val := 1

$F \rightarrow 2$
F.val := 2

$F \rightarrow 3$
F.val := 3

$F \rightarrow 4$
F.val := 4

$F \rightarrow 5$
F.val := 5

$F \rightarrow 6$
F.val := 6

$F \rightarrow 7$
F.val := 7

$F \rightarrow 8$
F.val := 8

$F \rightarrow 9$
F.val := 9

$F \rightarrow (E)$
F.val := E.val

Aqui são usados apenas atributos sintetizados:

- $A(E) = S(E) \cup H(E)$, com $S(E) = \{val\}$ e $H(E) = \{\}$
- $A(T) = S(T) \cup H(T)$, com $S(T) = \{val\}$ e $H(T) = \{\}$
- $A(F) = S(F) \cup H(F)$, com $S(F) = \{val\}$ e $H(F) = \{\}$

O tipo destes atributos ($E.val$, $T.val$ e $F.val$) é *nat* (de número natural), "+" indica a operação de soma e "*" indica a operação de multiplicação. A notação $E.val$ indica uma referência ao atributo *val* do símbolo não-terminal E.

Desta forma, as sentenças $1+2$, $3+4*5$ e $(3+4)*(5+6)$ produzem, respectivamente:

- $1+2$:
  3
- $3+4*5$:
  23
- $(3+4)*(5+6)$
  77

A figura a seguir ilustra uma árvore de derivação para a sentença $(3+4)*(5+6)$ devidamente decorada com os seus atributos avaliados.

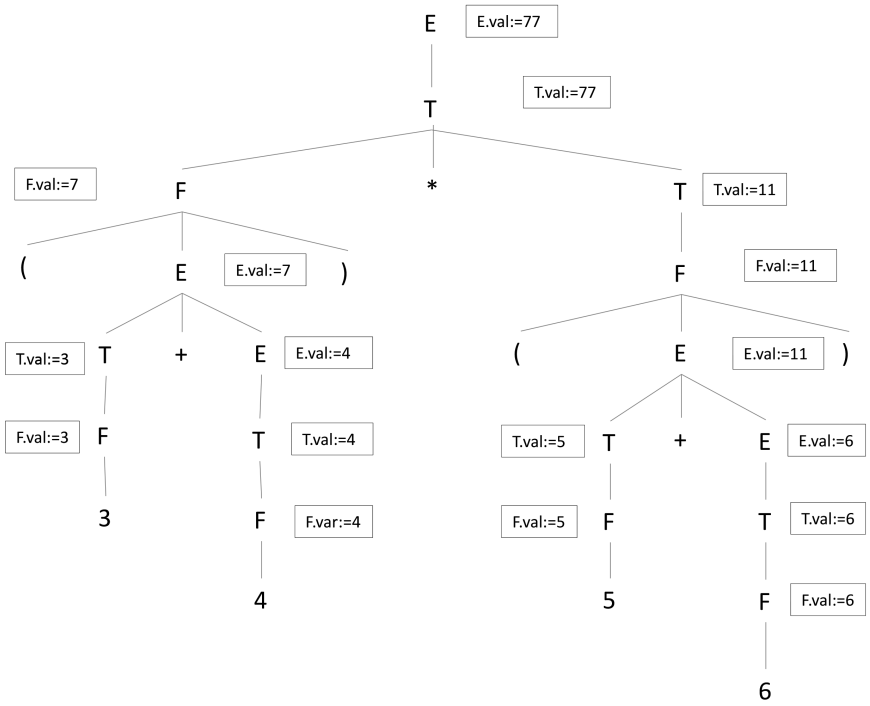

Figura 11: Árvore de derivação e atributos para $(3+4)*(5+6)$

Exemplo 2:
Considere novamente a gramática do Exemplo 4.7 que gera expressões aritméticas com soma, multiplicação e parênteses sobre o terminal a. Ela é modificada para usar a e b ao invés de apenas a e é apresentada a seguir:

$$\begin{aligned} E &\rightarrow T+E \\ E &\rightarrow T \\ T &\rightarrow F*T \\ T &\rightarrow F \end{aligned}$$

$$
\begin{array}{lcl}
F & \rightarrow & a \\
F & \rightarrow & b \\
F & \rightarrow & (E)
\end{array}
$$

Esta gramática gera sentenças como $a+a$, $a+a*b$ e $(a+a)*(b+b)$. A gramática de atributos apresentada a seguir transforma estas sentenças da notação infixa (em que os operadores são usados entre os operandos) para código-objeto de uma máquina de piha hipotética com as instruções PUSH (que insere o valor argumento no topo da pilha), ADD (que retira dois valores do topo da pilha substituindo-os pela soma deles) e MUL (que retira dois valores do topo da pilha substituindo-os pela multiplicação deles):

$E \rightarrow T+E$
E_1.code := T.code ++ E_2.code ++ 'ADD'

$E \rightarrow T$
E.code := T.code

$T \rightarrow F*T$
T_1.code := F.code ++ T_2.code ++ 'MUL'

$T \rightarrow F$
T.code := F.code

$F \rightarrow a$
F.code := 'PUSH a'

$F \rightarrow b$
F.code := 'PUSH b'

$F \rightarrow (E)$
F.code := E.code

Note, também neste caso, que todos os atributos são sintetizados ($E.code$, $T.code$ e $F.code$). O tipo destes atributos é *string* (ou cadeia de caracteres) e "++" indica a operação de concatenação de cadeias. Cadeias de caracteres (ou *strings*) literais são demilitadas por um apóstrofe simples (').

- $A(E) = S(E) \cup H(E)$, com $S(E) = \{code\}$ e $H(E) = \{\}$
- $A(T) = S(T) \cup H(T)$, com $S(T) = \{code\}$ e $H(T) = \{\}$
- $A(F) = S(F) \cup H(F)$, com $S(F) = \{code\}$ e $H(F) = \{\}$

Desta forma, as sentenças $a+a$, $a+a*b$ e $(a+a)*(b+b)$ produzem, respectivamente, os seguintes códigos:

- $a+a$:
  PUSH a
  PUSH a
  ADD
- $a+a*b$:
  PUSH a
  PUSH a
  PUSH b
  MUL
  ADD
- $(a+a)*(b+b)$:
  PUSH a
  PUSH a

ADD
PUSH b
PUSH b
ADD
MUL

Como ilustração, apresentamos a seguir a árvore de derivação para a sentença $(a+a)*(b+b)$ devidamente decorada com atributos e seus respectivos valores.

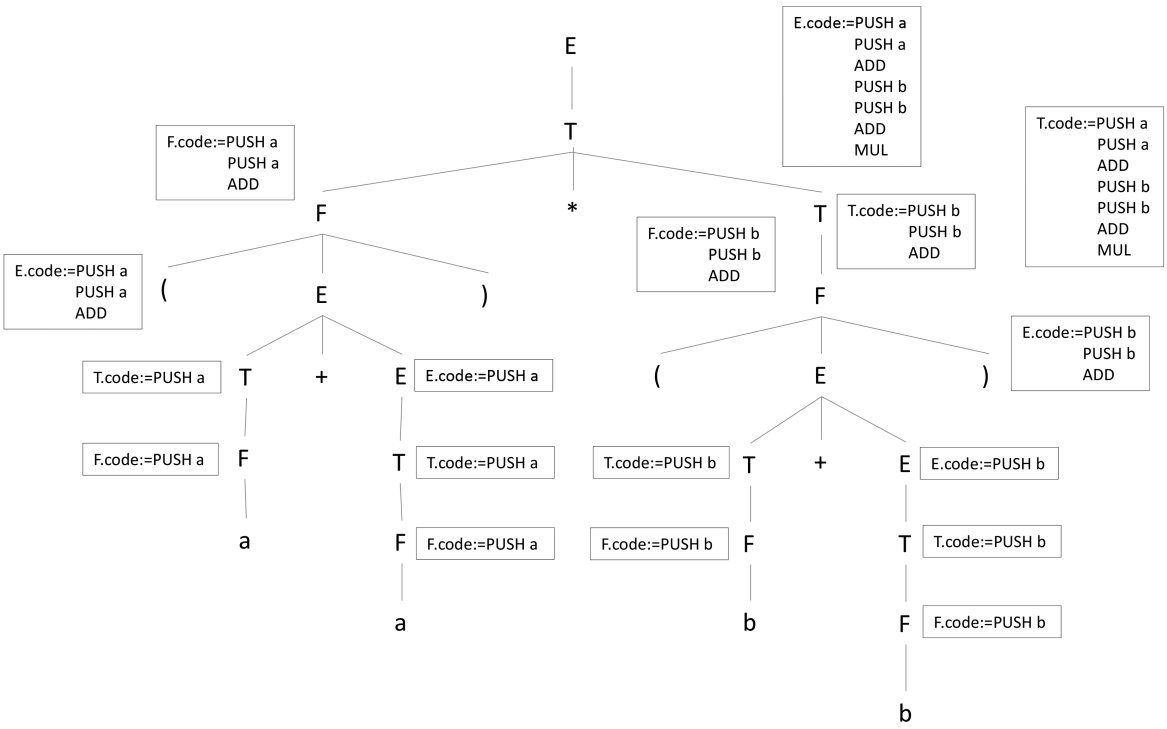

Figura 12: Árvore de derivação e atributos para $(a+a)*(b+b)$

Exemplo 3:
Considere a gramática livre de contexto do Exemplo 5.3, reproduzida a seguir:

$$
\begin{aligned}
Programa &\rightarrow Declaracoes\ Comandos \\
Declaracoes &\rightarrow Declaracoes\ Declaracao \\
Declaracoes &\rightarrow \varepsilon \\
Declaracao &\rightarrow \text{“\%”}\ Identificador \\
Comandos &\rightarrow Comandos\ Comando \\
Comandos &\rightarrow \varepsilon \\
Comando &\rightarrow \text{“\#”}\ Identificador\ \text{“}=\text{”}\ Expressao \\
Expressao &\rightarrow Expressao\ \text{“}+\text{”}\ Expressao \\
Expressao &\rightarrow Expressao\ \text{“}*\text{”}\ Expressao \\
Expressao &\rightarrow Identificador \\
Identificador &\rightarrow \text{“}a\text{”} \\
Identificador &\rightarrow \text{“}b\text{”}
\end{aligned}
$$

$$Identificador \quad \rightarrow \quad \text{“}c\text{”}$$

É exemplo de sentença pertencente a esta linguagem (as variáveis a e b são declaradas e depois usadas no comando):

$$\%a$$
$$\%b$$
$$\#a = a + b$$

A gramática apresentada, no entanto, como é sabido, não é capaz de representar as dependências de contexto da linguagem, a saber:

- Um mesmo nome (a, b ou c) não pode ser declarado mais de uma vez.
- Todos os nomes usados em um comando (tanto do lado esquerdo quanto do lado direito) devem ter sido previamente declarados.

Ela permite, por exemplo, que as seguintes cadeias sejam geradas:

- A variável a é duplamente declarada.

$$\%a$$
$$\%a$$
$$\#a = a + b$$

- A variável c não é declarada.

$$\%a$$
$$\%b$$
$$\#a = b + c$$

Uma gramática de atributos, tendo esta gramática livre de contexto como subjacente, pode ser construída com o objetivo de garantir que as dependências de contexto da linguagem sejam observadas.

Diferentemente dos exemplos anteriores, a gramática de atributos apresentada a seguir possui atributos sintetizados e herdados, além de alguns predicados:

- $A(Declaracoes) = S(Declaracoes) \cup H(Declaracoes)$,
  com $S(Declaracoes) = \{tab\}$ e $H(Declaracoes) = \{\}$
- $A(Declaracao) = S(Declaracao) \cup H(Declaracao)$,
  com $S(Declaracao) = \{tab\}$ e $H(Declaracao) = \{\}$
- $A(Identificador) = S(Identificador) \cup H(Identificador)$,
  com $S(Identificador) = \{val\}$ e $H(Identificador) = \{\}$
- $A(Comandos) = S(Comandos) \cup H(Comandos)$,
  com $S(Comandos) = \{\}$ e $H(Comandos) = \{tab\}$
- $A(Comando) = S(Comando) \cup H(Comando)$,
  com $S(Comando) = \{\}$ e $H(Comando) = \{tab\}$
- $A(Expressao) = S(Expressao) \cup H(Expressao)$,
  com $S(Expressao) = \{\}$ e $H(Expressao) = \{tab\}$

Observe a introdução de três predicados. O primeiro exige que os identificadores declarados sejam únicos. Ou seja, que toda nova variável declarada não faça parte do conjunto das variáveis declaradas até aquele ponto. O segundo exige que toda variável usada no lado esquerdo de um comando tenha sido previamente declarada. O terceiro, que toda variável usada em uma expressão tenha sido previamente declarada.

$$
\begin{array}{rcl}
\mathit{Programa} & \rightarrow & \mathit{Declaracoes}\ \mathit{Comandos} \\
 & & \boxed{\text{Comandos.tab} := \text{Declaracoes.tab}} \\
\mathit{Declaracoes} & \rightarrow & \mathit{Declaracoes}\ \mathit{Declaracao} \\
 & & \boxed{\text{Declaracoes\_1.tab} := \text{Declaracoes\_2.tab} \cup \text{Declaracao.tab}} \\
 & & \boxed{\text{Declaracao.tab} \not\subset \text{Declaracoes\_2.tab}} \\
\mathit{Declaracoes} & \rightarrow & \varepsilon \\
 & & \boxed{\text{Declaracoes.tab} := \emptyset} \\
\mathit{Declaracao} & \rightarrow & \text{“\%”}\ \mathit{Identificador} \\
 & & \boxed{\text{Declaracao.tab} := \text{Identificador.val}} \\
\mathit{Comandos} & \rightarrow & \mathit{Comandos}\ \mathit{Comando} \\
 & & \boxed{\text{Comando.tab} := \text{Comandos\_1.tab}} \\
 & & \boxed{\text{Comandos\_2.tab} := \text{Comandos\_1.tab}} \\
\mathit{Comandos} & \rightarrow & \varepsilon \\
\mathit{Comando} & \rightarrow & \text{“\#”}\ \mathit{Identificador}\ \text{“}=\text{”}\ \mathit{Expressao} \\
 & & \boxed{\text{Expressao.tab} := \text{Comando.tab}} \\
 & & \boxed{\text{Identificador.val} \subset \text{Comando.tab}} \\
\mathit{Expressao} & \rightarrow & \mathit{Expressao}\ \text{“}+\text{”}\ \mathit{Expressao} \\
 & & \boxed{\text{Expressao\_2.tab} := \text{Expressao\_1.tab}} \\
 & & \boxed{\text{Expressao\_3.tab} := \text{Expressao\_1.tab}} \\
\mathit{Expressao} & \rightarrow & \mathit{Expressao}\ \text{“}*\text{”}\ \mathit{Expressao} \\
 & & \boxed{\text{Expressao\_2.tab} := \text{Expressao\_1.tab}} \\
 & & \boxed{\text{Expressao\_3.tab} := \text{Expressao\_1.tab}} \\
\mathit{Expressao} & \rightarrow & \mathit{Identificador} \\
 & & \boxed{\text{Identificador.val} \subset \text{Expressao.tab}} \\
\mathit{Identificador} & \rightarrow & \text{“}a\text{”} \\
 & & \boxed{\text{Identificador.val} := \{\text{'a'}\}} \\
\mathit{Identificador} & \rightarrow & \text{“}b\text{”} \\
 & & \boxed{\text{Identificador.val} := \{\text{'b'}\}} \\
\mathit{Identificador} & \rightarrow & \text{“}c\text{”} \\
 & & \boxed{\text{Identificador.val} := \{\text{'c'}\}}
\end{array}
$$

Observe, na gramática acima, que, para cada regra, existem tantas funções de avaliação de atributos quanto sejam os atributos sintetizados do lado esquerdo da mesma e os atributos herdados do lado direito da mesma.

Essencialmente, esta gramática de atributos produz uma tabela de símbolos (na forma de um conjunto, de baixo para cima) com todos os nomes declarados até um certo ponto. Depois, esta tabela é repassada para o lado direito da árvore (onde ficam os comandos, de cima para baixo) para verificar se os nomes usados fazem parte da tabela de símbolos (e portanto foram previamente declarados). Todos os atributos possuem tipo *conjunto de caracteres*.

Desta forma, as cadeias $\%a\%b\#a = a + b$, $\%a\%a\#a = a + b$ e $\%a\%b\#a = b + c$ produzem, respectivamente, as seguintes árvores decoradas:

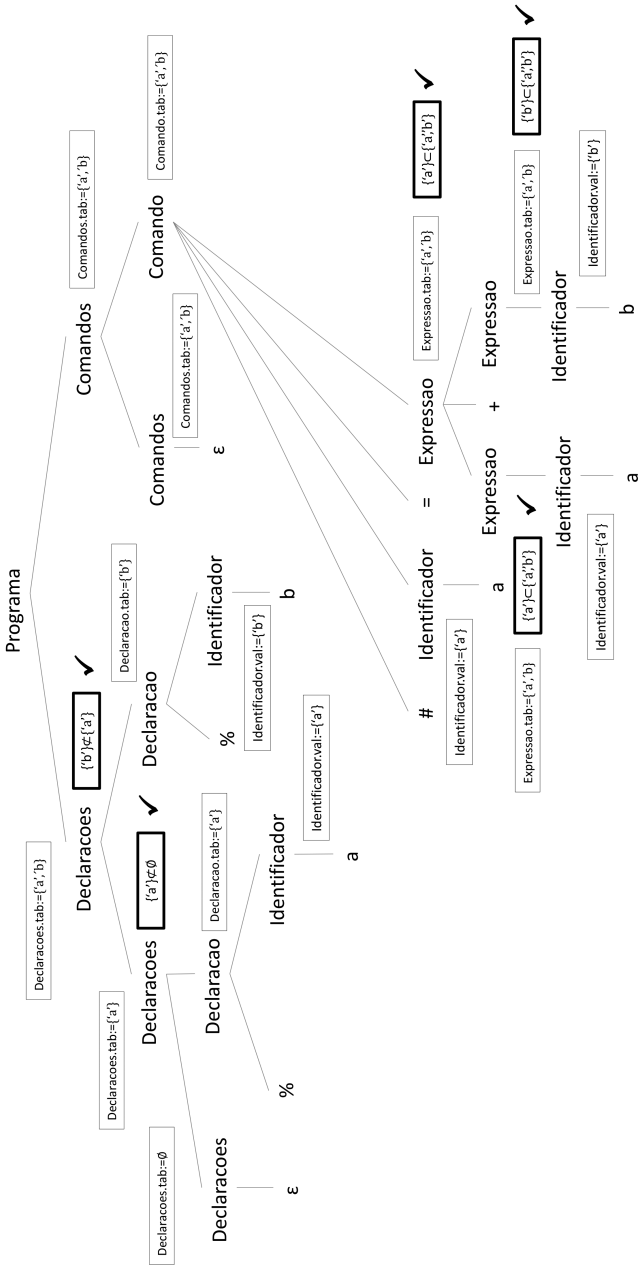

Figura 13: Árvore de derivação e atributos para $\%a\%b\#a = a + b$

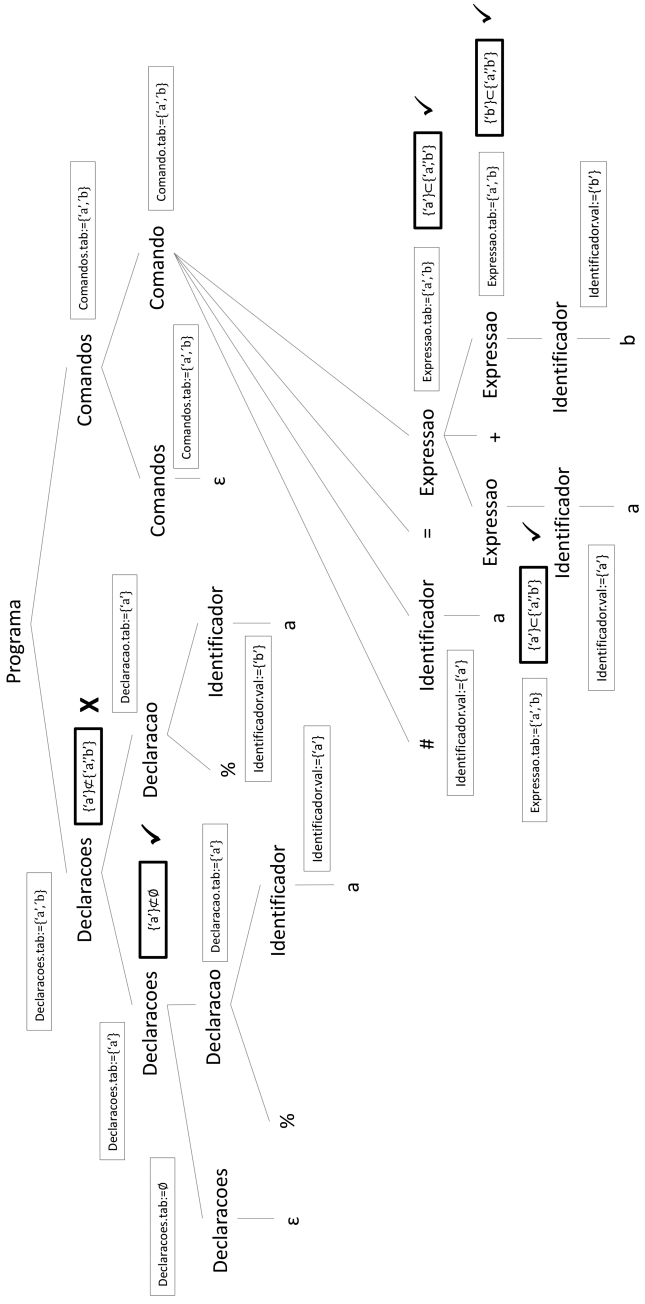

Figura 14: Árvore de derivação e atributos para $\%a\%a\#a = a + b$

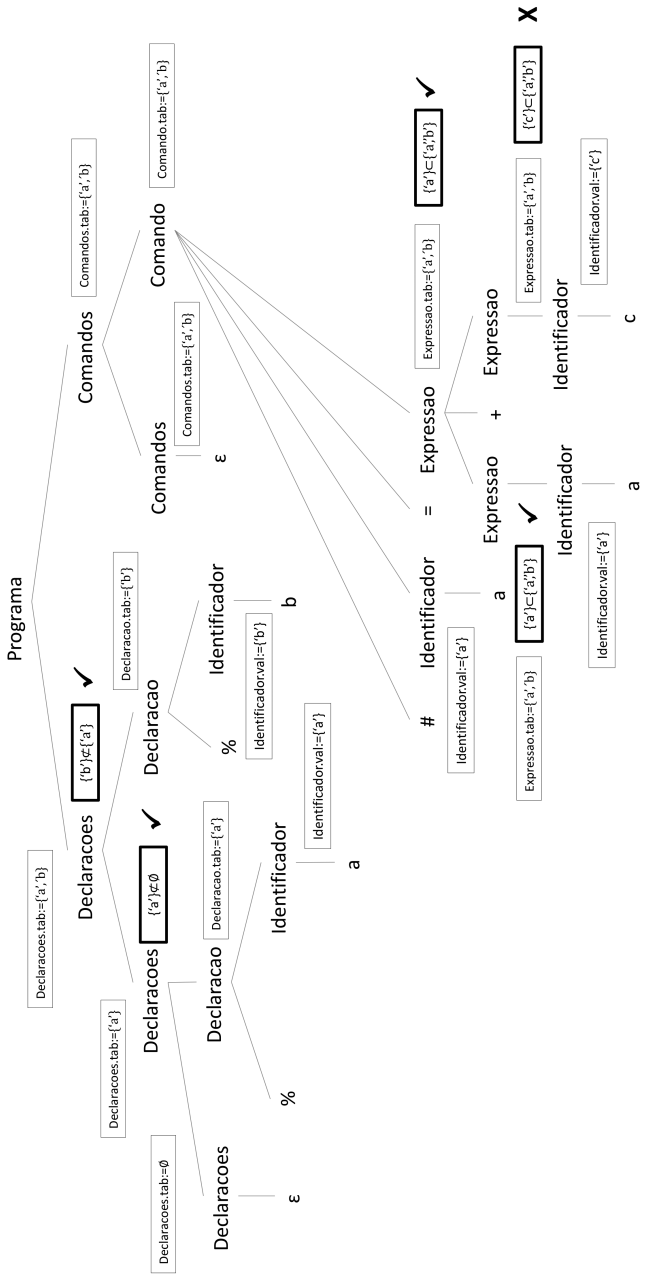

Figura 15: Árvore de derivação e atributos para $\%a\%b\#a = a + c$

Note que as árvores são decoradas não apenas com os valores dos atributos determinados pelas funções de avaliação dos atributos, como nos exemplos anteriores, mas também com o resultado da avaliação dos predicados. No primeiro caso, todos os predicados produzem VERDADEIRO (denotado por ✓). No segundo e terceiro casos, há um predicado em cada que resulta FALSO (denotado por **X**), causando a rejeição da entrada.

Os Exemplos 1 e 2 ilustram o uso de uma gramática de atributos para representar a semântica (ou seja, o significado) de uma sentença. No Exemplo 1 é usada a chamada “semântica denotacional", pois o significado de uma sentença é representado pelo valor que é calculado durante a sua geração. No Exemplo 2 é usada a chamada “semântica operacional", em que o significado de uma sentença é traduzido em termos das instruções de uma máquina mais simples. O Exemplo 3 ilustra o uso de uma gramática de atributos para representar as dependências de contexto (portanto sintaxe) da linguagem, sem ter nenhuma relação com a semântica (significado) das suas sentenças. Na

prática, gramáticas de atributos são usadas para as duas finalidades.

Normalmente o compilador cria um grafo de dependência dos atributos, de forma que todos eles possam todos ser avaliados corretamente, em alguma ordem (desde que não existam circularidades). Os atributos podem ser avaliados durante o curso da própria análise sintática ou, alternativamente, pode-se gerar uma árvore com uma representação intermediária do programa e decorar a mesma com os atributos de cada nó. Só então, numa fase posterior, os atributos são avaliados. No Capítulo 14 de citar fischer1988 discute-se em detalhes a questão da ordem de avaliação dos atributos, criando como consequência uma taxonomia de gramáticas de atributos.

Outra tentativa de superar as limitações das gramáticas livres de contexto, por meio de extensões destas, na representação das dependências de contexto da linguagem-fonte, são as gramáticas-W (citar wijngaarden1965, chastellier1969, koster1974), também conhecidas como gramáticas de van Wijngaarden (o seu inventor) ou ainda gramáticas de dois níveis. Elas recebem este nome pois cada definição consiste em duas gramáticas, sendo que uma elas gera as regras que são usadas pela segunda. Gramáticas-W foram usadas pela primeira vez na definição formal da sintaxe da linguagem de programação Algo-68 (citar wijngaarden1981). Mais informações, com exemplos, podem ser encontradas em citar slonneger1995 (Capítulo 4).

Gramáticas de atributos e gramáticas-W servem, portanto, para a representação formal das dependências de contexto e da semântica de linguagens de programação. Elas foram bastante usadas na década de 1960, mas deixaram de ser usadas desde então. No entanto, elas (principalmente as gramáticas de atributos) introduziram conceitos que ainda hoje são relevantes para a construção de compiladores.

49. `Página 455:`
`Inserir, depois do parágrafo` "*Linguagens sensíveis ao contexto...*"
`o parágrafo:`
"*Uma Máquina de Turing com fita limitada é, por definição, não determinística, mas também é possível definir uma versão determinística dela (modificando a função de transição de maneira correspondente). Entretanto, não se sabe ainda se as Máquinas de Turing com fita limitada não determinísticas reconhecem a mesma classe de linguagens que as Máquinas de Turing com fita limitada determinísticas (citar linz2011, citar wiki-lba).*"

50. `Página 552:`
`Teorema 7.7:`
`Inserir, depois de` "L_U, L_P e L_K":
", respectivamente Teoremas 6.5, 6.6 e 6.7
`Inserir, depois de` "que as aceitam":
" (respectivamente Teoremas 7.8, 7.9 e 7.10)"

51. `Página 591:`
Glossário
`Acrescentar os seguintes termos:`

Algoritmo CYK Algoritmo que verifica se uma dada cadeia de símbolos é gerada por uma gramática livre de contexto (qualquer). O nome deriva das iniciais dos seus inventores, Cocke, Yunger e Kasami.

Contradição Proposição que é sempre falsa, qualquer que seja a atribuição de valores às variáveis da mesma. Exemplo: $x \wedge \neg x$.

Gramática de atributos Gramática livre de contexto estendida com atributos (variáveis), funções de avaliação de atributos e eventualmente predicados. Permite a representação formal das dependências de contexto e da semântica das linguagens de programação.

Lema de Ogden Versão mais forte do *Pumping Lemma* para as linguagens livres de contexto. É usado, principalmente, para provar que uma linguagem não é livre de contexto.

Tautologia Proposição que é sempre verdadeira, qualquer que seja a atribuição de valores às variáveis da mesma. Exemplo: $x \vee \neg x$.

52. `Página 602:`
`Índice remissivo:`
`Acrescentar:`

“concatenação de cadeias, **65**”
“associatividade, 65”
“comutatividade, 65”
“elemennto neutro, 65”
“comprimento, 65”

53. Página 602:
    Índice remissivo:
    Substituir “concatenação” por “concatenação de linguagens, **68**”.

54. Página 605:
    Índice remissivo:
    Substituir “reversão” por “reversa”.

55. Página 607:
    Índice remissivo:
    Acrescentar:
    “reverso de cadeia, **65**”
    “idempotência, 65”
    “cadeia de cadeias, 66”

56. Página 607:
    Índice remissivo:
    Acrescentar:
    “reverso de linguagem, **72**”

```
@inproceedings{sakai1961,
title = "Syntax in universal translation",
author = "Sakai, Itiroo",
booktitle = "Proceedings of the International Conference on Machine Translation and Applie
month = "5-8 " # sep,
year = "1961",
address = "National Physical Laboratory, Teddington, UK",
}

@book{linz2011,
author = {Linz, Peter},
title = {An Introduction to Formal Languages and Automata},
year = {2011},
isbn = {9781449615529},
publisher = {Jones and Bartlett Publishers, Inc.},
address = {USA},
edition = {5th},
}

@misc{wiki-lba,
  author = {Wikipedia},
  title = {Linear bounded automaton},
  howpublished = "\url{https://en.wikipedia.org/wiki/Linear_bounded_automaton}",
  note = "[Online; accessado em 01-05-2024]"
}

@book{loehr2019,
  title={An Introduction to Mathematical Proofs},
  author={Loehr, N.A.},
```

```
  isbn={9780367338237},
  series={Textbooks in Mathematics Series},
  year={2019},
  publisher={CRC Press/Taylor \& Francis Group}
}

@article{knuth1968,
  author = {Knuth, Donald E.},
  journal = {{Mathematical Systems Theory}},
  number = {{2}},
  pages = {127-145},
  title = {{Semantics of Context-Free Languages}},
  volume = {{2}},
  year = {{1968}}
}

@inproceedings{knuth1990,
author = {Knuth, Donald E.},
title = {The Genesis of Attribute Grammars},
year = {1990},
isbn = {3540531017},
publisher = {Springer-Verlag},
address = {Berlin, Heidelberg},
booktitle = {Proceedings of the International Conference WAGA on Attribute Grammars and Th
pages = {1-12},
numpages = {12}
}

@book{aho2006,
  author = {Aho, Alfred V. and Lam, Monica S. and Sethi, Ravi and Ullman, Jeffrey D.},
  isbn = {0321486811},
  month = {August},
  edition = {2nd},
  publisher = {{Addison Wesley}},
  title = {Compilers: Principles, Techniques, and Tools (2nd Edition)},
  year = 2006
}

@inproceedings{johnson1978,
  title={Yacc: Yet Another Compiler-Compiler},
  author={S. C. Johnson},
  year={1978},
}

@book{fischer1988,
author = {Fischer, Charles N. and LeBlanc, Richard J.},
title = {Crafting a compiler},
year = {1988},
isbn = {0805332014},
publisher = {Benjamin-Cummings Publishing Co., Inc.},
address = {USA}
}
```

```
@book{slonneger1995,
author = {Slonneger, Kenneth and Kurtz, Barry},
title = {Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach
year = {1995},
isbn = {0201656973},
publisher = {Addison-Wesley Longman Publishing Co., Inc.},
address = {USA},
edition = {1st},
}

@book{sebesta2012,
author = {Sebesta, Robert W.},
title = {Concepts of Programming Languages},
year = {2012},
isbn = {0273769103},
publisher = {Pearson},
edition = {10th},
}

@inproceedings{chastellier1969,
  author       = {Guy de Chastellier and Alain Colmerauer},
  title        = {W-grammar},
  booktitle    = {Proceedings of the 24th national conference, {ACM} 1969, USA, 1969},
  pages        = {511--518},
  publisher    = {{ACM}},
  year         = {1969},
}

@unpublished{wijngaarden1965,
+ title  = {Orthogonal design and description of a formal language},
 author = {van Wijngaarden, Adriaan},
 number = {MR 76/65},
 year   = 1965,
 month  = jan
}

@article{wijngaarden1981,
author = {van Wijngaarden, A.},
title = {Revised Report of the Algorithmic Language Algol 68},
year = {1981},
issue_date = {1981},
publisher = {Computer History Museum},
address = {Mountain View, CA, USA},
number = {Sup 47},
issn = {0084-6198},
journal = {ALGOL Bull.},
month = aug,
pages = {1-119},
numpages = {119}
}

@book{waite1984,
author = {Waite, William and Goos, Gerhard},
```

```
year = {1984},
month = {01},
pages = {},
title = {Compiler Construction},
isbn = {0-387-90821-8},
}

@article{irons1961,
author = {Irons, Edgar T.},
title = {A syntax directed compiler for ALGOL 60},
year = {1961},
issue_date = {Jan. 1961},
publisher = {Association for Computing Machinery},
address = {New York, NY, USA},
volume = {4},
number = {1},
issn = {0001-0782},
journal = {Commun. ACM},
month = jan,
pages = {51-55},
numpages = {5}
}

@Inbook{koster1974,
author="Koster, C. H. A.",
editor="Bauer, F. L. and Eickel, J.",
title="Two-Level Grammars",
bookTitle="Compiler Construction: An Advanced Course",
year="1974",
publisher="Springer Berlin Heidelberg",
address="Berlin, Heidelberg",
pages="146--156",
isbn="978-3-662-21549-4",
}

@article{knuth1964,
author = {Knuth, Donald E.},
title = {Backus Normal Form vs. Backus Naur form},
year = {1964},
issue_date = {Dec. 1964},
publisher = {Association for Computing Machinery},
address = {New York, NY, USA},
volume = {7},
number = {12},
issn = {0001-0782},
journal = {Commun. ACM},
month = dec,
pages = {735-736},
numpages = {2}
}

@article{ogden1968,
  title={A helpful result for proving inherent ambiguity},
```

```
  author={William F. Ogden},
  journal={Mathematical systems theory},
  year={1968},
  volume={2},
  pages={191-194},
}

@article{wise1976,
title = {A strong pumping lemma for context-free languages},
journal = {Theoretical Computer Science},
volume = {3},
number = {3},
pages = {359-369},
year = {1976},
issn = {0304-3975},
author = {David S. Wise},
}

@book{shallit2008,
author = {Shallit, Jeffrey},
title = {A Second Course in Formal Languages and Automata Theory},
year = {2008},
isbn = {0521865727},
publisher = {Cambridge University Press},
address = {USA},
edition = {1},
}
```