Linguagens Livres de Contexto

Prof. Marcus Vinícius Midena Ramos

Universidade Federal do Vale do São Francisco

11 de novembro de 2021

marcus.ramos@univasf.edu.br
www.univasf.edu.br/~marcus.ramos

Bibliografia

1. *Linguagens Formais: Teoria, Modelagem e Implementação*
M.V.M. Ramos, J.J. Neto e I.S. Vega
Bookman, 2009

Roteiro

Linguagens de programação

- A maior aplicação das gramáticas livres de contexto ocorre na formalização sintática das linguagens de programação de alto nível;
- Rigorosamente, não se pode dizer que tais linguagens sejam propriamente livres de contexto;
- Na verdade, a formalização completa de sua sintaxe exigiria dispositivos mais complexos, como é o caso das gramáticas do tipo 1, uma vez que a grande maioria de tais linguagens apresenta dependências de contexto que não podem ser representadas por gramáticas do tipo 2, ou livres de contexto.

Aninhamento

- A característica que torna as gramáticas livres de contexto especialmente adequadas à formalização sintática das linguagens de programação é a sua capacidade de representação de **construções aninhadas**, que são freqüentemente encontradas em linguagens dessa categoria;
- Construções aninhadas costumam ocorrer em linguagens de programação, por exemplo, na construção de expressões aritméticas, em que subexpressões são delimitadas, através do uso de parênteses; na estruturação do fluxo de controle, em que comandos internos são inseridos como parte integrante de outros externos; na estruturação do programa, em que blocos, módulos, procedimentos e funções são empregados para criar diferentes escopos, etc.

Definição

Uma gramática livre de contexto é uma quádrupla (V, Σ, P, S) com os seguintes componentes:

- V: conjunto (finito e não-vazio) dos símbolos terminais e não-terminais;
- Σ: conjunto (finito e não-vazio) dos símbolos terminais; corresponde ao alfabeto da linguagem definida pela gramática;
- P: conjunto (finito e não-vazio) das regras de produção, todas no formato $\alpha \rightarrow \beta$, com $\alpha \in (V - \Sigma)$ e $\beta \in V^*$;
- S: raiz da gramática, $S \in (V - \Sigma)$.

Exemplo

Exemplo 1.1

Seja P o conjunto de regras abaixo e considerem-se Σ, V e S subentendidos.

$$
\begin{aligned}
\{E &\rightarrow T+E, && (1)\\
E &\rightarrow T, && (2)\\
T &\rightarrow F*T, && (3)\\
T &\rightarrow F, && (4)\\
F &\rightarrow (E), && (5)\\
F &\rightarrow a\} && (6)
\end{aligned}
$$

Exemplo

Considere-se a sentença $a * (a + a)$. Ela pode ser obtida através da seguinte seqüência de derivações:

$E \Rightarrow T \Rightarrow F * T \Rightarrow a * T \Rightarrow a * F \Rightarrow a * (E) \Rightarrow a * (T + E) \Rightarrow a * (F + E)$
$\Rightarrow a * (a + E) \Rightarrow a * (a + T) \Rightarrow a * (a + F) \Rightarrow a * (a + a)$

correspondente à aplicação das produções
$4.2, 4.3, 4.6, 4.4, 4.5, 4.1, 4.4, 4.6, 4.2, 4.4, 4.6$, nesta ordem.

Exemplo

Observe-se que a linguagem gerada pela gramática deste exemplo compreende as sentenças que representam expressões aritméticas corretamente formadas sobre o operando *a* com os operadores "$*$" e "$+$". Subexpressões delimitadas através de parênteses também são admitidas, e podem ser compostas com base nas mesmas regras utilizadas para construir a expressão inicial. Não fazem parte da linguagem definida por esta gramática, por exemplo, cadeias em que não haja plena correspondência do símbolo "$($" com seu par "$)$". Em outras palavras, trata-se de uma linguagem que admite o aninhamento de expressões através do uso de parênteses como delimitadores.

Exercícios

Obter gramáticas livres de contexto que gerem cada uma das seguintes linguagens:

1. $\{a^i b^i \mid i \geq 0\}$;
2. $\{a^i b^i \mid i \geq 1\}$;
3. $\{a^i b^j \mid i \leq j\}$;
4. $\{a^i b^j \mid i < j\}$;
5. $\{a^i b^j \mid i \geq j\}$;
6. $\{a^i b^j \mid i > j\}$;
7. $\{a^i b^j \mid i \neq j\}$;
8. $\{a^i b^i \mid i$ é par$\}$;
9. $\{a^i b^j \mid i$ é par e j é ímpar;
10. $\{a^i b^{2i} \mid i > 0\}$;

Exercícios

Obter gramáticas livres de contexto que gerem cada uma das seguintes linguagens:

11. $\{a^{2i}b^{i} \mid i > 0\}$;
12. $\{a^{2i}b^{3i} \mid i > 0\}$;
13. $\{a^{2i}b^{3i+1} \mid i > 0\}$;
14. $\{a^{2i+2}b^{3i} \mid i > 0\}$;
15. $\{a^{2i+2}b^{3i} \mid i \geq 0\}$;
16. $\{a^{i}b^{i}a^{j}b^{j} \mid i \geq 0, j \geq 0\}$;
17. $\{a^{i}b^{j}a^{j}b^{i} \mid i \geq 0, j \geq 0\}$;
18. $\{(a^{i}b^{i})^{*} \mid i \geq 0\}$;
19. $\{a^{i}b^{i}c^{*} \mid i \geq 0\}$;
20. $\{a^{i}c^{*}b^{i} \mid i \geq 0\}$;

Exercícios

Obter gramáticas livres de contexto que gerem cada uma das seguintes linguagens:

21. $\{a^i c^* b^i \mid i \geq 1\}$;
22. $\{c^* a^i b^i \mid i \geq 0\}$;
23. $\{c^* a^i c^* b^i c^* \mid i \geq 0\}$;
24. $\{(ab)^i (ba)^i \mid i \geq 0\}$;
25. $\{a^i (bbb)^i \mid i \geq 0\}$;
26. $\{(a^*)^i (b^*)^i \mid i \geq 0\}$;
27. $\{a^i (b^*)^i \mid i \geq 0\}$;
28. $\{(a^i b^i)^k (a^j b^j)^k \mid i \geq 0, j \geq 0, k \geq 0\}$;
29. $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$;
30. $\{w \in \{a,b\}^* \mid |w|_a \neq |w|_b\}$;

Linguagens estritamente livres de contexto

- O aninhamento de construções é a característica que distingue as linguagens estritamente do tipo 2 das linguagens do tipo 3: pertencem a uma linguagem estritamente do tipo 2 apenas e tão somente as sentenças em que, para cada ocorrência de um dado delimitador (no caso, o “abre-parênteses”), haja em correspondência a ocorrência de um outro, através do qual se forma o par (no caso, o “fecha-parênteses”);
- Além disso, a subcadeia situada entre esse par de delimitadores pode ser gerada através das mesmas regras de formação válidas para a sentença completa, caracterizando dessa maneira o aninhamento de suas construções.

Aninhamentos sintáticos

- Através de regras de produção livres de contexto, é possível especificar construções mais restritas do que as obteníveis mediante o uso de produções de gramáticas lineares;
- Com regras livres de contexto é possível caracterizar aninhamentos sintáticos, os quais não podem ser descritos apenas com produções lineares à direita ou à esquerda (ou qualquer combinação delas);

Aninhamentos sintáticos

- Gramáticas livres de contexto permitem impor restrições adicionais àquelas que se podem construir em gramáticas regulares, podendo assim caracterizar subconjuntos das linguagens regulares que gozem da propriedade determinada pelos aninhamentos sintáticos;
- Gramáticas livres de contexto tornam-se muito úteis para a especificação de linguagens de programação, a maioria das quais exibe aninhamentos sintáticos.

Não-terminal auto-recursivo central

Um não-terminal Y é dito “self-embedded”, ou **auto-recursivo central**, se, a partir dele, for possível derivar alguma forma sentencial em que o não-terminal Y ressurja, delimitado por cadeias não-vazias de terminais à sua esquerda e à sua direita:

$$Y \Rightarrow^* \alpha Y \beta, \quad \text{com } \alpha, \beta \in \Sigma^+$$

Gramática auto-embutida

Um não-terminal Z é dito simplesmente **auto-recursivo** se, a partir dele, for possível derivar alguma forma sentencial em que Z ressurja, acompanhado de pelo menos uma cadeia não-vazia de terminais à sua esquerda ou à sua direita:

$$Z \Rightarrow \alpha Z \beta, \quad \text{com } \alpha, \beta \in \Sigma^*, \alpha\beta \neq \varepsilon$$

Se uma gramática livre de contexto G possui pelo menos um não-terminal auto-recursivo central, diz-se que G é **auto-embutida** (do inglês “self-embedded”).

Não-terminal auto-recursivo central essencial

Um símbolo não-terminal **essencial** é aquele que não pode ser removido da gramática (ou substituído) sob pena de provocar modificações na linguagem sendo definida. Uma linguagem L é dita **estritamente livre de contexto**, ou livre de contexto não-regular, se e apenas se todas as gramáticas que geram L forem auto-embutidas, ou seja, se todas elas possuírem pelo menos um não-terminal **auto-recursivo central essencial**.

Não-terminal auto-recursivo central essencial

O simples fato de uma gramática ser auto-embutida não garante a não-regularidade da linguagem definida: é possível identificar linguagens regulares geradas por gramáticas com não-terminais auto-recursivos centrais que, nesses casos, não são essenciais. O Exemplo 1.2 ilustra essa situação.

Exemplo

Exemplo 1.2
A gramática cujas regras constituem o conjunto abaixo é do tipo 2 e possui um não-terminal auto-recursivo central (S, em decorrência da produção $S \rightarrow aSa$), podendo portanto ser caracterizada como uma gramática auto-embutida:

$$\begin{aligned} \{S &\rightarrow aS, \\ S &\rightarrow bS, \\ S &\rightarrow a, \\ S &\rightarrow b, \\ S &\rightarrow aSa\} \end{aligned}$$

Exemplo

No entanto, a linguagem gerada por essa gramática é $\{a,b\}^+$, ou seja, a linguagem é regular. Na verdade, é fácil observar que essa linguagem também pode ser gerada por um conjunto de regras equivalente, em que a última produção da gramática acima é removida:

$$\begin{aligned} \{S &\rightarrow aS, \\ S &\rightarrow bS, \\ S &\rightarrow a, \\ S &\rightarrow b\} \end{aligned}$$

Tal fato ocorre, neste caso particular, porque a produção $S \rightarrow aSa$ é **não-essencial** à gramática, ou seja, a sua inclusão no conjunto P de produções em nada contribui para modificar a linguagem definida pelas demais produções.

Linguagem estritamente livre de contexto

- Quando todas as alternativas de substituição para um símbolo não-terminal são não-essenciais, diz-se que o símbolo em questão é **não-essencial**;
- Se ele for um não-terminal auto-recursivo central, diz-se que o não-terminal auto-recursivo central é não-essencial;
- Não-terminais auto-recursivos centrais que caracterizam gramáticas auto-embutidas podem ou não ser essenciais;
- Se houver pelo menos um não-terminal auto-recursivo central essencial em alguma gramática, a correspondente linguagem é dita **estritamente livre de contexto**;
- Se todos os não-terminais auto-recursivos centrais de uma gramática forem não-essenciais, a linguagem por ela definida é regular.

Propriedade do auto-embutimento

- A “Self-embedding Property” exprime a capacidade que têm as gramáticas livres de contexto para representarem aninhamentos sintáticos;
- Se $Y \Rightarrow^* \alpha Y \beta$, então, da aplicação sucessiva dessa seqüência de derivações, resulta que $\alpha\alpha Y \beta\beta$, $\alpha\alpha\alpha Y \beta\beta\beta$ etc. são formas sentenciais que geram sentenças pertencentes a $L(G)$;
- A cada subcadeia α é possível impor a existência de uma outra subcadeia β que estabeleça a correspondência com α;
- Gramáticas regulares não exibem a “Self-embedding Property”, sendo esta uma propriedade característica das gramáticas livres de contexto não-regulares.

Exemplo

Exemplo 1.3
Considere-se a linguagem das expressões aritméticas definida no Exemplo 1.1. Observe-se que $E \Rightarrow T \Rightarrow F \Rightarrow (E)$, ou seja, $E \Rightarrow^* (E)$.
Logo, E é um não-terminal auto-recursivo central, e a gramática a que ele pertence é auto-embutida. É possível demonstrar que todas as gramáticas que geram essa linguagem são auto-embutidas, o que caracteriza a linguagem como não-regular.
Note-se que, como $E \Rightarrow^* (E)$, a cada abre-parênteses que antecede a subexpressão E corresponde sempre um fecha-parênteses logo após E. Note-se também que todas as sentenças derivadas das formas sentenciais $({}^iE)^i$, $i \geqslant 1$, também pertencem à linguagem definida, sendo obtidas pela aplicação repetida da seqüência de derivações apresentada.

Análise sintática

- Quando se consideram linguagens especificadas através de gramáticas livres de contexto, deve-se também considerar de que forma é feita a aceitação sintática de suas sentenças para fins de compilação e/ou interpretação;
- Quando se trata de efetuar o reconhecimento de sentenças, o que se busca, na verdade, é localizar uma seqüência de produções que, quando aplicada à raiz da gramática, forneça como resultado, através da série correspondente de derivações, a sentença fornecida para análise;
- Sendo possível completar a derivação, diz-se que a sentença pertence à linguagem; caso contrário, que ela não pertence à linguagem.

Análise sintática

- No entanto, para cada sentença pertencente à linguagem definida através de uma gramática livre de contexto, é geralmente possível identificar uma grande quantidade de seqüências distintas de derivação, resultado de escolhas arbitrárias do particular símbolo não-terminal a ser substituído em cada passo da derivação, todas elas resultando na mesma sentença analisada;
- Na prática, no entanto, costuma-se fixar alguns critérios de derivação de sentenças (mais à direita e mais à esquerda), para permitir a construção e a operação sistemáticas dos reconhecedores sintáticos.

Metalinguagens

- Como conseqüência do elevado interesse prático despertado pelas gramáticas livres de contexto, inúmeras notações foram desenvolvidas para facilitar a formalização sintática das linguagens artificiais, permitindo criar definições formais mais legíveis e concisas do que as obtidas usualmente com a notação algébrica.
- Tais notações, denominadas **metalinguagens**, permitem a representação de linguagens livres de contexto, sendo equipotentes, portanto, à notação algébrica introduzida anteriormente. A primeira e talvez mais importante delas é a BNF, abreviatura de “Backus-Naur Form”.

BNF

Na notação BNF, os não-terminais são representados por textos delimitados pelos metassímbolos “<” e “>”; para distingui-los dos símbolos terminais, o metassímbolo “$\rightarrow$” é substituído por “$::=$” e, finalmente, todas as alternativas de substituição para um mesmo não-terminal são agrupadas, separando-se umas das outras com o metassímbolo “$|$”. Os terminais são denotados sem delimitadores.

Exemplo

Exemplo 2.1
Em BNF, a gramática que representa expressões aritméticas aninháveis, sobre operandos *a*, com os operadores adição e multiplicação, anteriormente apresentada no Exemplo 1.1, torna-se:

$$\begin{aligned} <E> &::= <T> + <E> \mid <T> \\ <T> &::= <F> * <T> \mid <F> \\ <F> &::= a \mid (<E>) \end{aligned}$$

BNF

Empregada pela primeira vez no início da década de 1960, na publicação que definiu formalmente a linguagem de programação Algol 60, a notação BNF se tornou extremamente popular justamente por ter demonstrado na prática, pela primeira vez, a viabilidade de uso de uma notação formal para a representação da sintaxe de uma linguagem de programação, representando um dos primeiros resultados práticos advindos do interesse dos profissionais de computação pelas então recém descobertas gramáticas livres de contexto.

Metalinguagens

Embora largamente utilizada ainda hoje, a notação BNF enfrenta atualmente a “concorrência” de outras metalinguagens igualmente importantes, como é o caso de vários de seus dialetos, da Notação de Wirth, dos Diagramas de Sintaxe, das Expressões Regulares Estendidas e de inúmeras outras. Apesar disso, a notação algébrica continuará sendo empregada preferencialmente neste texto por ser a mais adequada ao estudo teórico e conceitual das linguagens formais. Eventualmente, poderá ser empregada a BNF, em exemplos de linguagens de programação.

BNF estendida

- Expressões regulares foram definidas como uma notação bastante concisa e adequada para a representação de linguagens regulares;
- Uma importante extensão das expressões regulares são as expressões regulares estendidas, que, em combinação com a BNF, disponibilizam as vantagens do uso de expressões regulares para a classe das linguagens livres de contexto, constituindo assim uma metalinguagem alternativa para a representação das mesmas;
- Essa metalinguagem, denominada **BNF estendida** (ou **EBNF**), resulta da fusão das definições da BNF e da expressão regular.

Expressão regular estendida e BNF estendida

- Uma **expressão regular estendida** é, por definição, uma expressão regular que admite como operandos os símbolos não-terminais da gramática, em adição aos terminais;
- Um conjunto de regras gramaticais representado através da notação **BNF estendida** é um conjunto de expressões regulares estendidas, cada uma delas associada a um símbolo não-terminal distinto.

Exemplo

Exemplo 2.2
Considere-se a gramática livre de contexto:

$$G = (\{S,X,Y,Z,a,b,c,d,e,f,g\},\{a,b,c,d,e,f,g\},P,S)$$

com P apresentado a seguir, que gera uma linguagem L estritamente livre de contexto.

$$\begin{aligned} P = \{S &\rightarrow XYZ \\ S &\rightarrow g \\ X &\rightarrow aX \\ X &\rightarrow a \\ Y &\rightarrow Sb \\ Z &\rightarrow cdZ \\ Z &\rightarrow eZ \\ Z &\rightarrow f\} \end{aligned}$$

Exemplo

Transcrito para a BNF, este conjunto de produções resulta:

$$
\begin{aligned}
<S> &::= <X><Y><Z> \mid g \\
<X> &::= a<X> \mid a \\
<Y> &::= <S>b \\
<Z> &::= cd<Z> \mid e<Z> \mid f
\end{aligned}
$$

Exemplo

Não é difícil perceber que $<X>$ gera cadeias compostas por um ou mais símbolos a. Então, a regra $<X> ::= a<X> \mid \varepsilon$ pode ser substituída pela regra $<X> ::= aa^*$. De maneira análoga, a regra $<Z> ::= cd<Z> \mid e<Z> \mid f$ pode ser substituída por $<Z> ::= (cd \mid e)^* f$. Note-se, em ambos os casos, a substituição do uso de símbolo não-terminal no lado direito das regras, pelo uso do operador fechamento reflexivo e transitivo ($*$) para representar a repetição de termos.

Exemplo

O novo conjunto de regras torna-se, portanto:

$$
\begin{aligned}
<S> &::= <X><Y><Z> \mid g \\
<X> &::= aa^* \\
<Y> &::= <S>b \\
<Z> &::= (cd \mid e)^* f
\end{aligned}
$$

Exemplo

A substituição das definições dos símbolos <X>, <Y> e <Z> na regra do símbolo <S> resulta em:

$$<S> \quad ::= \quad aa^*<S>b(cd \mid e)^*f \mid g$$

- O lado direito da regra acima ($aa^*<S>b(cd \mid e)^*f \mid g$) é, como se pode perceber, uma expressão regular estendida, pois possui a forma de uma expressão regular, acrescida da referência ao símbolo não-terminal <S>. O lado esquerdo e o lado direito, juntamente, constituem uma regra gramatical representada na notação BNF estendida;
- A regra acima explicita a presença de um símbolo não-terminal auto-recursivo essencial em G (no caso, o símbolo <S>), suficiente para caracterizar L como sendo livre de contexto e não-regular. Em termos informais, L pode também ser representada como $(aa^*)^n g(b(cd \mid e)^*f)^n$, com $n \geqslant 0$.

Vantagens

- A principal vantagem decorrente uso da BNF estendida, em comparação com a notação algébrica, ou mesmo com a BNF, resulta da possibilidade de representar a repetição de formas sintáticas sem a necessidade de definições gramaticais recursivas (aquelas em que o símbolo não-terminal que estiver sendo definido ressurge, direta ou indiretamente, em formas sentenciais derivadas do mesmo), substituindo-as pela definição de iterações explícitas (através do uso do operador fechamento reflexivo e transitivo);
- Isso proporciona um entendimento mais fácil da linguagem por ela definida, sendo ainda útil em determinados métodos de construção de reconhecedores sintáticos a partir de gramáticas livres de contexto.

Não-terminais auto-recursivas centrais essenciais

- Nem sempre uma gramática livre de contexto poderá ser reduzida a uma única regra em BNF estendida, ainda que esta represente uma definição recursiva;
- A quantidade mínima de regras a que se pode chegar é igual à quantidade de símbolos não-terminais auto-recursivos centrais essenciais que a gramática possui, uma vez que cada um destes é responsável pela incorporação de uma característica distinta de aninhamento sintático à linguagem definida pela gramática, não podendo ser removidos da gramática sem prejuízo da linguagem por ela definida.

Gramáticas com uma única regra

- Gramáticas livres de contexto que representam linguagens regulares poderão sempre ser manipuladas até a eliminação de todos os símbolos não-terminais da gramática, exceto, naturalmente, a raiz, resultando em uma única expressão regular (não-estendida) que gera a linguagem definida pela gramática;
- Dessa maneira, a BNF estendida é um formalismo que une as vantagens das expressões regulares às da BNF, constituindo importante alternativa tanto para a representação de linguagens regulares quanto para a de linguagens livres de contexto.

Conceito

A representação da estrutura de sentenças ou formas sentenciais de linguagens livres de contexto, na forma de árvores bidimensionais, é um recurso muito utilizado, tanto na teoria quanto na prática da implementação de linguagens, uma vez que:

1. Proporciona meios para uma melhor visualização da estrutura das sentenças da linguagem, facilitando a análise das mesmas.
2. Auxilia na demonstração formal de teoremas, na interpretação de certos resultados teóricos e na assimilação de vários conceitos.
3. Facilita a representação interna, nos compiladores e interpretadores, da estrutura das sentenças analisadas, registrando importantes informações estruturais sobre as mesmas, a serem utilizadas em outros estágios do processamento da linguagem.

Definição

Formalmente, uma **árvore de derivação** é um sistema de representação de seqüências de derivações em uma gramática livre de contexto $G = (V, \Sigma, P, S)$, consistindo em um grafo orientado e ordenado, acíclico, com as seguintes propriedades:

1. Todo vértice é rotulado com um elemento de $V \cup \{\varepsilon\}$;
2. O rótulo da raiz é S;
3. Os rótulos de vértices internos são elementos de $N = V - \Sigma$;
4. Se um vértice tem o rótulo A, e $X_1, X_2, \ldots X_n$ são descendentes diretos de A, ordenados da esquerda para a direita, então $A \Rightarrow X_1, X_2, \ldots X_n$ deve pertencer ao conjunto P de regras;
5. Se um vértice possui o rótulo ε, então este vértice deve ser simultaneamente uma folha e descendente único de seu ancestral direto.

Fronteira de uma árvore

- Define-se como **fronteira de uma árvore** a cadeia formada pela concatenação dos símbolos correspondentes aos rótulos dos vértices que são folhas, considerados no sentido da esquerda para a direita, e de cima para baixo;
- Essa cadeia de símbolos (terminais e/ou não-terminais) corresponde à forma sentencial cuja estrutura está sendo representada pela árvore.

Exemplo

Exemplo 3.1
Considere-se a gramática das expressões aritméticas anteriormente apresentada no Exemplo 1.1. A estrutura da sentença $a * (a + a)$, de acordo com essa gramática, pode ser representada pela árvore de derivação da Figura 1.

Exemplo

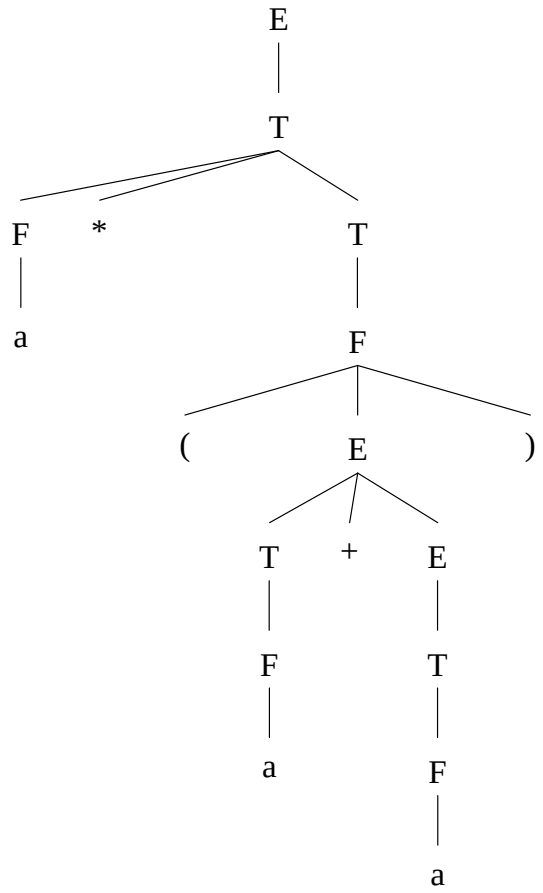

Figura 1: Árvore de derivação para $E \Rightarrow^* a * (a + a)$

Árvores e derivações

Naturalmente, nem toda árvore necessita ter como raiz S, nem tampouco possuir como fronteira $\alpha \in \Sigma^*$, $S \Rightarrow^* \alpha$. É comum que se considerem subárvores cuja raiz seja um elemento X qualquer, $X \in N$, e a fronteira, uma cadeia $\gamma \in V^*$, tal que $X \Rightarrow^* \gamma$.

Exemplo

Exemplo 3.2
A derivação não-trivial $T \Rightarrow^* a^*(E)$, segundo a gramática do Exemplo 1.1, pode ser representada através da subárvore da Figura 2.

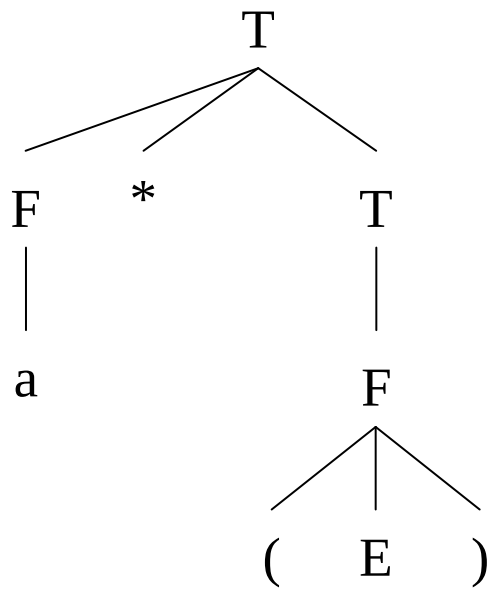

Figura 2: Árvore de derivação para $T \Rightarrow^* a^*(E)$

Árvores e derivações

- A seguir, será apresentado o teorema fundamental das árvores de derivação;
- Ele estabelece a existência de pelo menos uma árvore de derivação para toda e qualquer cadeia derivável a partir da raiz da gramática, e também que toda e qualquer fronteira de uma árvore corretamente construída sobre uma gramática livre de contexto G corresponde a uma cadeia derivável a partir da raiz de G.

Árvores e derivações

Teorema 3.1 "Seja $G = (V, \Sigma, P, S)$ uma gramática livre de contexto. Então $S \Rightarrow^* \alpha$ se e apenas se existir uma árvore de derivação sobre G com fronteira α."

Por se tratar de um teorema bastante intuitivo, não será apresentada a sua prova formal, que no entanto pode ser encontrada em livros da área. A importância deste teorema se deve ao fato de que ele "credencia" as árvores de derivação como uma representação válida da estrutura das formas sentenciais geradas por uma gramática.

Informações que as árvores não representam

- Árvores de derivação não contêm informação sobre a particular seqüência em que foram aplicadas as produções para a obtenção de uma dada forma sentencial: elas informam apenas **quais** foram as produções aplicadas, mas não **em que ordem**;
- Assim, para cada árvore de derivação, é possível identificar diversas seqüências distintas de derivação que resultem na mesma forma sentencial: basta alterar a ordem em que os não-terminais são substituídos em cada passo da derivação.

Derivações mais à esquerda e mais à direita

- Diz-se que a derivação de uma forma sentencial em uma gramática livre de contexto é uma **derivação mais à esquerda**, quando a substituição de um não-terminal pelo lado direito de uma produção que o define é feita substituindo-se sempre o não-terminal que ocorre mais à esquerda na cadeia que representa a forma sentencial em questão;
- De maneira análoga, uma **derivação mais à direita** é aquela em que é sempre o não-terminal mais à direita, na forma sentencial, que é substituído pela sua definição.

Exemplo

Exemplo 4.1
Considere-se a gramática do Exemplo 1.3. A seguir são apresentadas duas seqüências de derivação para a sentença $a + a$. Na primeira, todas as derivações são mais à esquerda, e na segunda, mais à direita. Note-se que as seqüências de produções utilizadas em cada caso são distintas, e também que diversas outras seqüências poderiam ser obtidas combinando-se arbitrariamente os não-terminais a serem substituídos em cada forma sentencial.

Exemplo

1. Derivações com substituições mais à esquerda apenas:

   $E \Rightarrow T+E \Rightarrow F+E \Rightarrow a+E \Rightarrow a+T \Rightarrow a+F \Rightarrow a+a$

   Produções aplicadas: $4.1, 4.4, 4.6, 4.2, 4.4, 4.6$

2. Derivações com substituições mais à direita apenas:

   $E \Rightarrow T+E \Rightarrow T+T \Rightarrow T+F \Rightarrow T+a \Rightarrow F+a \Rightarrow a+a$

   Produções aplicadas: $4.1, 4.2, 4.4, 4.6, 4.4, 4.6$

3. Derivações com substituições de diversos tipos:

   $E \Rightarrow T+E \Rightarrow F+E \Rightarrow F+T \Rightarrow a+T \Rightarrow a+F \Rightarrow a+a$

   Produções aplicadas: $4.1, 4.4, 4.2, 4.6, 4.4, 4.6$

Gramática não-ambígua

Diz-se que uma gramática livre de contexto é **não-ambígua** se, para toda e qualquer cadeia pertencente à linguagem por ela gerada, existir uma única seqüência de derivações mais à esquerda e uma única seqüência de derivações mais à direita que a geram.

Gramática ambígua

Diz-se que uma gramática livre de contexto é **ambígua** quando existir pelo menos uma cadeia, pertencente à linguagem por ela gerada, que possua mais de uma seqüência distinta de derivações, feitas exclusivamente através de substituições de não-terminais mais à esquerda ou mais à direita. Se houver mais de uma seqüência com substituições mais à esquerda, então haverá também mais de uma com substituições mais à direita, conforme demonstrado no Teorema 4.1.

Múltiplas derivações à esquerda e à direita

Teorema 4.1 "Se $w \in L(G)$, com G sendo uma gramática livre de contexto, e existindo duas (ou mais) derivações mais à esquerda para w em G, então existem também, correspondentemente, duas (ou mais) derivações mais à direita para w em G."

Múltiplas derivações à esquerda e à direita

As gramáticas livres de contexto possuem a propriedade de que as derivações de cada um dos símbolos não-terminais presentes em uma mesma forma sentencial podem ser feitas de forma independente umas das outras. Se $S \Rightarrow^* \alpha X \beta \Rightarrow^* \alpha\gamma\beta$, então a derivação $X \Rightarrow \gamma$ independe de α e β.

Múltiplas derivações à esquerda e à direita

Considere-se uma gramática livre de contexto G e a seqüência de derivações mais à esquerda $S \Rightarrow^* \alpha_1 X \gamma$, com $\alpha_1 \in \Sigma^*, X \in N$, e $\gamma \in V^*$. Considere-se que $X \Rightarrow^* \alpha_2$, com $\alpha_2 \in \Sigma^*$ e $\gamma \Rightarrow^* \alpha_3$, com $\alpha_3 \in \Sigma^*$. Então, $S \Rightarrow^* \alpha_1 X \gamma \Rightarrow^* \alpha_1 \alpha_2 \alpha_3$ e $\alpha_1 \alpha_2 \alpha_3 \in L(G)$.

Múltiplas derivações à esquerda e à direita

Sejam $X \to \beta_1$ e $X \to \beta_2$ duas produções distintas de G. Então, existem duas derivações à esquerda distintas para a cadeia $\alpha_1\alpha_2\alpha_3$:

- $S \Rightarrow^* \alpha_1 X\gamma \Rightarrow \alpha_1\beta_1\gamma \Rightarrow^* \alpha_1\alpha_2\gamma \Rightarrow^* \alpha_1\alpha_2\alpha_3$
- $S \Rightarrow^* \alpha_1 X\gamma \Rightarrow \alpha_1\beta_2\gamma \Rightarrow^* \alpha_1\alpha_2\gamma \Rightarrow^* \alpha_1\alpha_2\alpha_3$

Múltiplas derivações à esquerda e à direita

Se $\alpha_1\alpha_2\alpha_3 \in L(G)$ e $X \Rightarrow^* \alpha_2$, então existe uma seqüência de derivações mais à direita $S \Rightarrow^* \mu X\alpha_3$, com $\mu \in V^*$ e, além disso, $\mu \Rightarrow^* \alpha_1$. Portanto, $S \Rightarrow^* \mu X\alpha_3 \Rightarrow^* \mu\alpha_2\alpha_3 \Rightarrow^* \alpha_1\alpha_2\alpha_3$.

Múltiplas derivações à esquerda e à direita

Como, por hipótese, $X \Rightarrow \beta_1 \Rightarrow^* \alpha_2$ e $X \Rightarrow \beta_2 \Rightarrow^* \alpha_2$, então existem duas seqüências de derivações mais à direita para a cadeia $\alpha_1\alpha_2\alpha_3$:

- $S \Rightarrow^* \mu X \alpha_3 \Rightarrow \mu\beta_1\alpha_3 \Rightarrow^* \mu\alpha_2\alpha_3 \Rightarrow^* \alpha_1\alpha_2\alpha_3$
- $S \Rightarrow^* \mu X \alpha_3 \Rightarrow \mu\beta_2\alpha_3 \Rightarrow^* \mu\alpha_2\alpha_3 \Rightarrow^* \alpha_1\alpha_2\alpha_3$

Múltiplas derivações à esquerda e à direita

De maneira análoga, é possível demonstrar que, para cada seqüência distinta de derivações mais à esquerda que geram uma mesma cadeia da linguagem, existe uma seqüência distinta de derivações mais à direita correspondente, que gera a mesma cadeia.

Exemplo

Exemplo 4.2
A gramática cujo conjunto de regras está abaixo apresentado é equivalente à gramática anteriormente utilizada na definição da linguagem das expressões aritméticas sobre $\{a, +, *, (,)\}$ do Exemplo 1.1. Diferentemente daquela, no entanto, apenas um símbolo não-terminal (E) é utilizado:

$$\begin{aligned} \{E &\rightarrow E + E, \\ E &\rightarrow E * E, \\ E &\rightarrow a, \\ E &\rightarrow (E)\} \end{aligned}$$

Exemplo

Adotando-se o critério das derivações mais à esquerda, pode-se perceber que a sentença $a + a * a$ pode ser derivada ao menos de duas formas distintas:

1. Aplicando-se inicialmente a produção $E \rightarrow E + E$:

   $E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$

2. Aplicando-se inicialmente a produção $E \rightarrow E * E$:

   $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$

Exemplo

Como conseqüência, a gramática anterior é ambígua. É possível provar, no entanto, que a gramática do Exemplo 1.1 (com os não-terminais E, T e F) admite uma única seqüência de derivação para cada sentença pertencente à linguagem. Note-se, em particular, que isso ocorre com a sentença acima considerada. Como conclusão, esta linguagem pode ser indistintamente definida através de uma gramática ambígua ou de uma gramática não-ambígua.

Exemplo

Exemplo 4.3

Considere-se o fragmento de gramática abaixo apresentado, que ilustra um problema típico de determinadas linguagens de programação de alto nível — a ambigüidade na construção de comandos condicionais aninhados.

$$
\begin{aligned}
\langle prog \rangle &\rightarrow \ldots \langle com \rangle \ldots \\
\langle com \rangle &\rightarrow \langle cond \rangle \\
\langle cond \rangle &\rightarrow \texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \\
\langle cond \rangle &\rightarrow \texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \texttt{ else } \langle com \rangle \\
\langle exp \rangle &\rightarrow \ldots
\end{aligned}
$$

Exemplo

Passa-se a analisar a estrutura da seguinte forma sentencial:

$$\texttt{if} \langle exp \rangle \texttt{ then if} \langle exp \rangle \text{ then } \langle com \rangle \texttt{ else } \langle com \rangle$$

Para se determinar uma eventual ambigüidade nessa forma sentencial, escolhe-se inicialmente e fixa-se um critério de substituição de não-terminais — por exemplo, as substituições dos não-terminais mais à esquerda. Neste caso, é fácil verificar que a forma sentencial acima admite duas seqüências distintas de derivação.

Exemplo

1. Primeira alternativa:

$$
\begin{aligned}
\langle prog \rangle &\Rightarrow \ldots \langle com \rangle \ldots \\
&\Rightarrow \ldots \langle cond \rangle \ldots \\
&\Rightarrow \ldots \texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \texttt{ else } \langle com \rangle \ldots \\
&\Rightarrow \ldots \texttt{if } \langle exp \rangle \texttt{ then } \langle cond \rangle \texttt{ else } \langle com \rangle \ldots \\
&\Rightarrow \ldots \texttt{if } \langle exp \rangle \texttt{ then if } \langle exp \rangle \texttt{ then } \langle com \rangle \\
&\quad \texttt{else } \langle com \rangle \ldots
\end{aligned}
$$

Exemplo

2. Segunda alternativa:

$$
\begin{aligned}
\langle prog \rangle &\Rightarrow \ldots \langle com \rangle \ldots \\
&\Rightarrow \ldots \langle cond \rangle \ldots \\
&\Rightarrow \ldots \texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \ldots \\
&\Rightarrow \ldots \texttt{if } \langle exp \rangle \texttt{ then } \langle cond \rangle \ldots \\
&\Rightarrow \ldots \texttt{if } \langle exp \rangle \texttt{ then if } \langle exp \rangle \texttt{ then } \langle com \rangle \\
&\quad \texttt{else } \langle com \rangle \ldots
\end{aligned}
$$

Exemplo

No primeiro caso, o ramo do "`else`" é considerado como parte integrante do comando condicional mais externo e, no segundo, este mesmo ramo é considerado como parte do comando condicional mais interno:

$$\underbrace{\texttt{if } \langle exp \rangle \texttt{ then } \underbrace{\texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle}\ \texttt{else } \langle com \rangle}$$

$$\underbrace{\texttt{if } \langle exp \rangle \texttt{ then } \underbrace{\texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \texttt{ else } \langle com \rangle}}$$

Assim, a forma sentencial apresentada é ambígua, o que também torna ambígua a gramática. Como se pode verificar facilmente, as mesmas conclusões poderiam também ter sido obtidas caso fossem consideradas apenas substituições mais à direita.

Eliminação de ambigüidades

Normalmente, ambigüidades costumam ser eliminadas através da utilização de construções gramaticais que impeçam a existência de mais de uma seqüência de derivações mais à esquerda (e, portanto também mais à direita) para cada sentença pertencente à linguagem, desde que fixado e observado *a priori*, naturalmente, um critério único para a substituição dos não-terminais nas formas sentenciais.

Exemplo

Exemplo 4.4

Seja a seguinte gramática, inspirada na do Exemplo 4.3:

$$
\begin{aligned}
\langle prog \rangle &\rightarrow \ldots \langle com \rangle \ldots \\
\langle com \rangle &\rightarrow \langle cond \rangle \\
\langle cond \rangle &\rightarrow \texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \texttt{ fi} \\
\langle cond \rangle &\rightarrow \texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \texttt{ else } \langle com \rangle \texttt{ fi} \\
\langle exp \rangle &\rightarrow \ldots
\end{aligned}
$$

Exemplo

A associação do "`else`" ao comando condicional mais externo ou mais interno é feita, de acordo com esta nova gramática, respectivamente:

$$\underbrace{\texttt{if } \langle exp \rangle \texttt{ then } \underbrace{\texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \texttt{ fi}}\texttt{ else } \langle com \rangle \texttt{ fi}}$$

ou

$$\underbrace{\texttt{if } \langle exp \rangle \texttt{ then } \underbrace{\texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \texttt{ else } \langle com \rangle \texttt{ fi}}\texttt{ fi}}$$

Linguagem livre de contexto ambígua

Diz-se que uma linguagem livre de contexto é **ambígua** se ela puder ser gerada por uma gramática livre de contexto ambígua, e **não-ambígua** se ela puder ser gerada por uma gramática livre de contexto não-ambígua. No entanto, raramente se costuma dizer que uma linguagem é ambígua, uma vez que uma mesma linguagem pode ser gerada por inúmeras gramáticas distintas (ambíguas e não-ambíguas).

Linguagem livre de contexto inerentemente ambígua

Há, no entanto, casos em que se pode provar, apesar de isso ser, em geral, uma tarefa consideravelmente complexa, que determinadas linguagens são **inerentemente ambíguas**, indicando com isso que toda e qualquer gramática que gera a linguagem em questão deve ser necessariamente ambígua. Em casos como esse, o conceito de ambigüidade pode ser estendido também para caracterizar mais um atributo da linguagem.

Exemplo

Exemplo 4.5
A linguagem $\{a^n b^n c^m d^m \mid n \geqslant 1, m \geqslant 1\} \cup \{a^n b^m c^m d^n \mid n \geqslant 1, m \geqslant 1\}$ é inerentemente ambígua. A demonstração pode ser encontrada em Hopcroft79.

Exemplo

Exemplo 4.6
A linguagem $\{a^i b^j c^k \mid i = j \text{ ou } j = k\}$ é inerentemente ambígua. A demonstração pode ser encontrada em Aho72.

Árvores e derivações canônicas

Teorema 4.2 "Seja G uma gramática livre de contexto. Para toda cadeia $w \in L(G)$, o número de seqüências distintas de derivações mais à esquerda (e portanto mais à direita) é igual ao número de árvores de derivação distintas que representam w."

Árvores e derivações canônicas

Prova:
Pode ser encontrada em Hopcroft79. Não é difícil, no entanto, intuir que, para cada forma sentencial obtida através de derivações mais à esquerda, a substituição do não-terminal mais à esquerda por produções distintas, sejam elas quantas forem, dá origem a uma quantidade idêntica de árvores distintas, as quais são diferenciadas pelos nós que representam os filhos do nó que representa esse não-terminal. De forma análoga, o raciocínio inverso também se aplica.

Árvores e derivações canônicas

Quando se consideram gramáticas livres de contexto não-ambíguas, a ordem em que são feitas as substituições dos símbolos não-terminais nas formas sentenciais é irrelevante do ponto de vista das cadeias que podem ser geradas e das respectivas árvores de derivação. Como mostra o Teorema 4.3, qualquer que seja a ordem de substituição dos não-terminais escolhida na geração de uma mesma cadeia, a árvore de derivação será sempre a mesma.

Árvores e derivações canônicas

Teorema 4.3 "Seja G uma gramática livre de contexto não-ambígua. Para toda cadeia $w \in L(G)$, toda e qualquer seqüência de derivações que produz w é representada através da mesma e única árvore de derivação."

Árvores e derivações canônicas

Se G é uma gramática livre de contexto não-ambígua, então, conforme a definição, toda e qualquer cadeia $w \in L(G)$ possui uma única seqüência de derivações mais à esquerda ou mais à direita, mesmo que existam outras seqüências não-canônicas de derivação para essa mesma cadeia. Conforme o Teorema 4.2, existe uma única árvore de derivação para w. Logo, todas as derivações de w em G são representadas por uma mesma árvore de derivação.

Árvores e derivações canônicas

Conforme a definição, uma gramática livre de contexto não-ambígua é aquela para a qual cada cadeia pertencente à linguagem gerada por tal gramática é gerada por uma única seqüência de derivações mais à esquerda e por uma única seqüência de derivações mais à direita. O Teorema 4.3 estende esse resultado, e estabelece que tais seqüências de derivações correspondem a uma mesma árvore de derivação.

Árvores e derivações canônicas

Em resumo: cada uma das cadeias pertencentes a uma linguagem gerada por uma gramática livre de contexto não-ambígua possui uma única seqüência de derivações mais à esquerda, uma única seqüência de derivações mais à direita e uma única árvore de derivação.

Árvores e derivações canônicas

Esta importante propriedade das linguagens livres de contexto (e portanto também das linguagens regulares) permite que se adote uma ordem arbitrária para a aplicação das regras gramaticais na derivação de uma sentença. Tecnologicamente, isso pode ser aproveitado escolhendo-se uma ordem que seja econômica do ponto de vista algorítmico.

Árvores e derivações canônicas

A existência de uma única seqüência canônica de derivações (uma mais à esquerda e outra mais à direita) para cada sentença de uma linguagem gerada por uma gramática não-ambígua é explorada nos algoritmos de construção de reconhecedores sintáticos determinísticos para linguagens livres de contexto, cuja operação consiste exatamente na busca sistemática de tal seqüência (e, conseqüentemente, da correspondente árvore de derivação), se essa seqüência existir, para cada cadeia de entrada que lhe seja submetida. Do ponto de vista tecnológico, esse resultado permite a busca da solução de implementação mais barata, conforme a linguagem considerada.

Gramática livre de contexto ambígua

Por outro lado, o Teorema 4.2 permite estender o conceito de gramática livre de contexto ambígua, que passa a ser caracterizado de três formas distintas, porém equivalentes entre si. Diz-se que uma gramática livre de contexto é ambígua se for possível identificar, na linguagem por ela gerada, pelo menos uma cadeia que:

- possa ser derivada por duas ou mais seqüências distintas de derivações mais à esquerda, ou
- possa ser derivada por duas ou mais seqüências distintas de derivações mais à direita, ou
- possa ser representada por duas ou mais árvores de derivação distintas.

Exemplo

Exemplo 4.7
Considere-se o fragmento de gramática utilizado para ilustrar o problema da ambigüidade em comandos condicionais (Exemplo 4.3). De acordo com essa gramática, a forma sentencial:

$$\texttt{if}\ \langle exp \rangle\ \texttt{then}\ \texttt{if}\ \langle exp \rangle\ \texttt{then}\ \langle com \rangle\ \texttt{else}\ \langle com \rangle$$

pode ser derivada segundo duas seqüências distintas de derivações mais à esquerda. Como conseqüência, existem também duas árvores de derivação distintas que possuem como fronteira essa mesma forma sentencial, conforme mostram as Figuras 3 e 4.

Exemplo

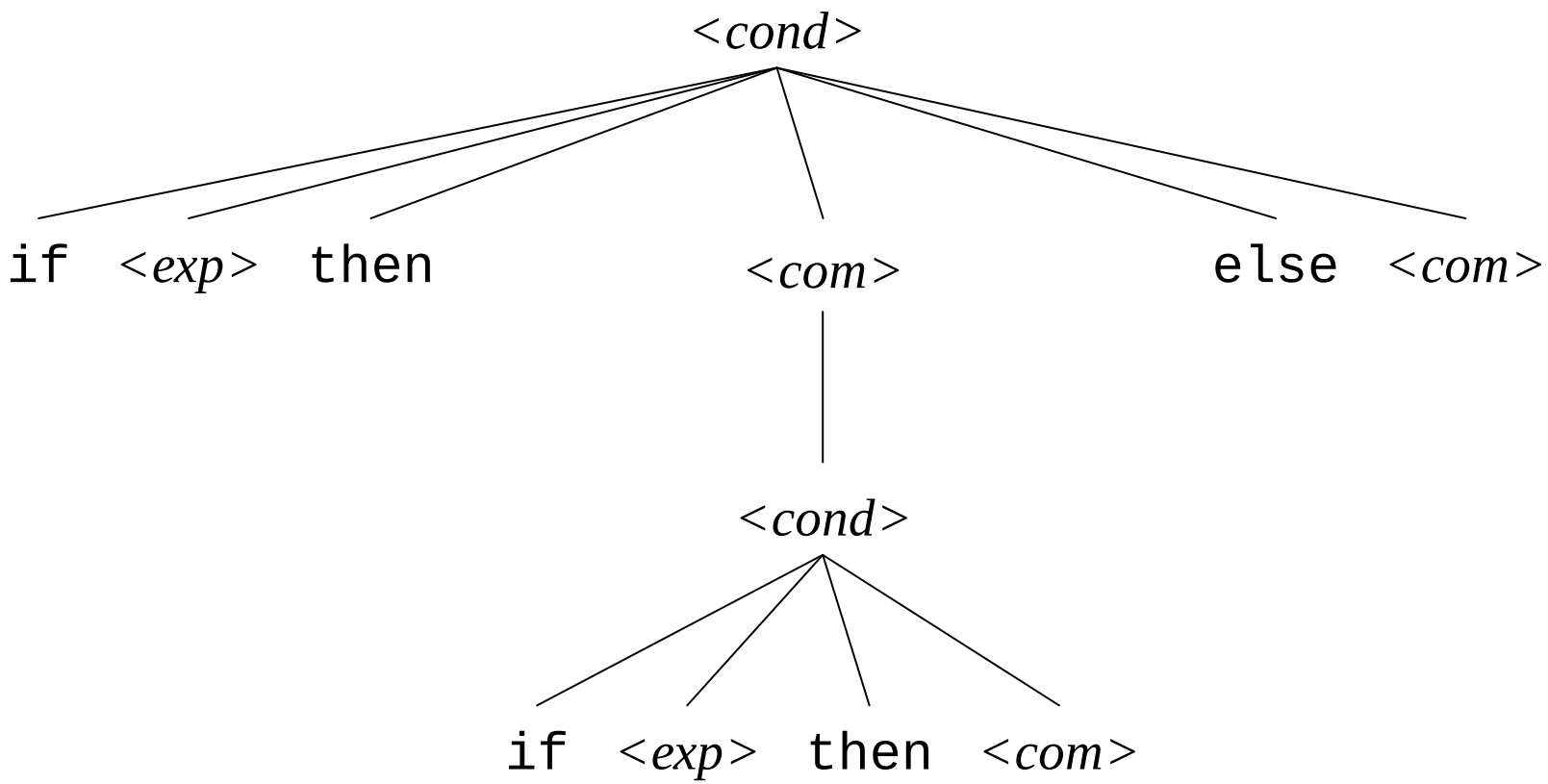

Figura 3: Ambigüidade no comando IF: associação interna

Exemplo

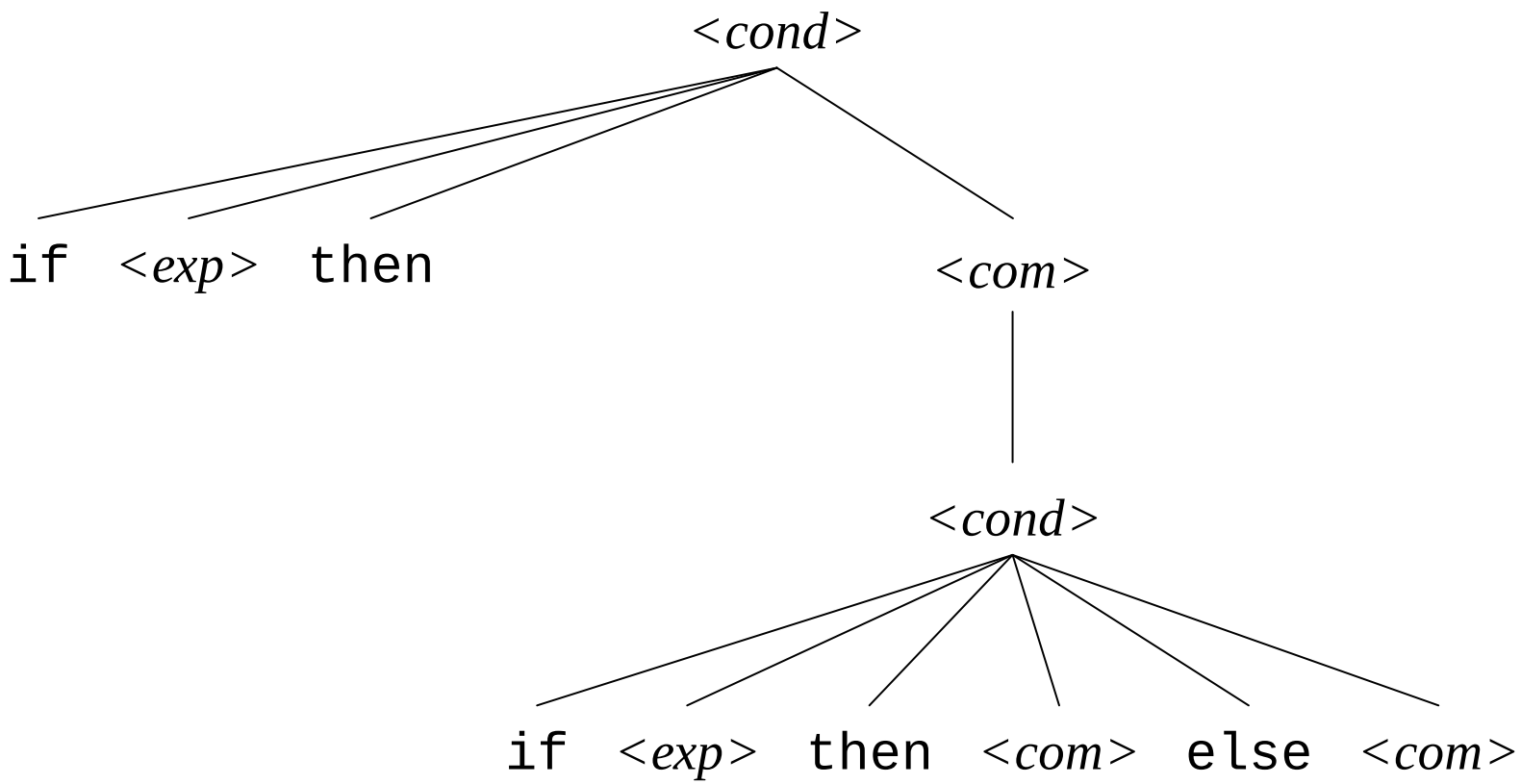

Figura 4: Ambigüidade no comando IF: associação externa

Etapas

Por uma questão de conveniência no estudo das gramáticas livres de contexto, muitas vezes é necessário submetê-las a simplificações que visam torná-las mais apropriadas para a demonstração de teoremas, para a verificação de propriedades, ou simplesmente para facilitar a sua análise.
Tais simplificações consistem em manipulações gramaticais que não afetam a linguagem definida pela gramática original, e são de três tipos (os novos termos serão definidos mais adiante):

1. Eliminação de símbolos inacessíveis e inúteis;
2. Eliminação de produções em vazio;
3. Eliminação de produções unitárias.

Símbolo inacessível

- Diz-se que um símbolo $Y \in V$ — terminal ou não-terminal — é **inacessível**, se não for possível derivar qualquer forma sentencial em que se registre alguma ocorrência de Y. Caso contrário, o símbolo é dito **acessível**. Formalmente, símbolos acessíveis Y são aqueles para os quais existem derivações da forma $S \Rightarrow^* \alpha Y \beta$, com $\alpha, \beta \in V^*$. Símbolos inacessíveis, portanto, são aqueles para os quais inexistem derivações com tais características.

Símbolo inútil

- Diz-se que um símbolo Y — não-terminal — é **inútil**, se não for possível derivar pelo menos uma cadeia formada exclusivamente por terminais (ou a cadeia vazia) a partir de Y. Caso contrário, o símbolo é dito **útil**. Formalmente, símbolos úteis Y são aqueles para os quais existem derivações da forma $Y \Rightarrow^* \gamma$, com $\gamma \in \Sigma^*$. Símbolos inúteis, portanto, são aqueles para os quais inexistem derivações com tais características. Cumpre notar que símbolos terminais são, por definição, sempre úteis.

Símbolo acessível e útil

Um símbolo Y que seja simultaneamente acessível e útil é, portanto, aquele que satisfaz à condição:

$$S \Rightarrow^* \alpha Y \beta \Rightarrow^* \alpha \gamma \beta, \quad \text{com } \alpha, \beta \in V^*, \gamma \in \Sigma^*$$

Em outras palavras, um símbolo Y é acessível e útil se e somente se:

1. Y está presente em pelo menos uma forma sentencial derivada a partir da raiz da gramática, e
2. Y deriva pelo menos uma cadeia pertencente ao conjunto Σ^* (condição verificada trivialmente pelos símbolos terminais).

Eliminação de símbolos inacessíveis e inúteis

A transformação de uma gramática em outra equivalente, isenta de símbolos inacessíveis e inúteis, pode ser feita em dois passos, através de dois algoritmos distintos: o primeiro algoritmo encarrega-se de eliminar da gramática original os símbolos inúteis; o segundo algoritmo elimina todos os símbolos inacessíveis.

Eliminação de símbolos inacessíveis e inúteis

Teorema 5.1 "Toda linguagem livre de contexto pode ser gerada por uma gramática livre de contexto em que não há símbolos inacessíveis ou inúteis."

A demonstração deste teorema pode ser realizada tomando-se como base os dois algoritmos apresentados a seguir. O Algoritmo 5.1 mostra como eliminar os não-terminais que não geram cadeias de terminais, bem como as produções em que os mesmos comparecem, preservando em N apenas os símbolos do conjunto $N' = \{Y \in N \mid Y \Rightarrow^* \alpha, \text{ com } \alpha \in \Sigma^*\}$.

Eliminação de símbolos inúteis

Algoritmo 5.1 *"Eliminação de símbolos inúteis em gramáticas livres de contexto."*

- *Entrada: uma gramática livre de contexto* $G = (V, \Sigma, P, S)$*, tal que* $L(G) \neq \emptyset$.
- *Saída: uma gramática livre de contexto* $G' = (V', \Sigma, P', S)$*, tal que* $L(G') = L(G)$ *e* $Y \in N'$ *se e apenas se* $L(Y) \neq \emptyset$.

Eliminação de símbolos inúteis

Método:

1. $N_0 \leftarrow \emptyset$;
2. $i \leftarrow 1$;
3. $N_i \leftarrow N_{i-1} \cup \{Y \mid Y \rightarrow \alpha \in P \text{ e } \alpha \in (N_{i-1} \cup \Sigma)^*\}$;
4. *Se* $N_i \neq N_{i-1}$, *então:*
   1. $i \leftarrow i+1$;
   2. *Desviar para (3);*

   Caso contrário:
   1. $N' \leftarrow N_i$;
   2. $P' \leftarrow \{A \rightarrow X_1 X_2 \ldots X_n \in P \mid A, X_1, X_2, \ldots, X_n \in (N_i \cup \Sigma)\}$.

Eliminação de símbolos inacessíveis

Algoritmo 5.2 *"Eliminação de símbolos inacessíveis em gramáticas livres de contexto."*

- *Entrada: uma gramática livre de contexto* $G = (V, \Sigma, P, S)$.
- *Saída: uma gramática livre de contexto* $G' = (V', \Sigma', P', S)$, *tal que* $L(G') = L(G)$ *e* $Y \in V'$ *se e apenas se* $\exists S \Rightarrow^* \alpha Y \beta$, *com* $\alpha, \beta \in V^*$.

Eliminação de símbolos inacessíveis

Método:

1. $V_0 \leftarrow \{S\}$;
2. $i \leftarrow 1$;
3. $V_i \leftarrow V_{i-1} \cup \{X_j \in V, 1 \leqslant j \leqslant n \mid A \rightarrow X_1 X_2 \ldots X_n \in P \text{ e } A \in V_{i-1}\}$;
4. *Se* $V_i \neq V_{i-1}$, *então:*
   1. $i \leftarrow i + 1$;
   2. *Desviar para (3);*

   Caso contrário:
   1. $N' \leftarrow V_i \cap N$;
   2. $\Sigma' \leftarrow V_i \cap \Sigma$;
   3. $P' \leftarrow \{A \rightarrow X_1 X_2 \ldots X_n \in P \mid A, X_1, X_2 \ldots X_n \in V_i\}$.

Exemplo

Exemplo 5.1
Considere-se $G = (V, \Sigma, P, S)$, com:

$$\begin{aligned} V &= \{S, A, B, C, a, b\} \\ \Sigma &= \{a, b\} \\ P &= \{S \to A \mid B, A \to aB \mid bS \mid b, B \to AB \mid Ba, C \to AS \mid b\} \end{aligned}$$

Aplicando-se o algoritmo de eliminação de não-terminais que não geram cadeias de terminais, obtém-se:

$$\begin{aligned} N_0 &= \emptyset \\ N_1 &= \{A, C\} \\ N_2 &= \{A, C, S\} \\ N_3 &= \{A, C, S\} \end{aligned}$$

Exemplo

Continuação

Logo, $G' = \{V', \Sigma, P', S\}$, com:

$$\begin{aligned} V' &= \{S, A, C, a, b\} \\ P' &= \{S \to A, A \to bS \mid b, C \to AS \mid b\} \end{aligned}$$

Note-se que o não-terminal B foi eliminado de N uma vez que não é possível derivar qualquer cadeia de terminais a partir dele. Aplicando-se a G' o Algoritmo 5.2, para a eliminação de símbolos inacessíveis, obtém-se:

$$\begin{aligned} V_0 &= \{S\} \\ V_1 &= \{S, A\} \\ V_2 &= \{S, A, b\} \\ V_3 &= \{S, A, b\} \end{aligned}$$

Exemplo

Continuação

e $G'' = \{V'', \Sigma'', P'', S\}$, com:

$$\begin{aligned} V'' &= \{S, A, b\} \\ \Sigma'' &= \{b\} \\ P'' &= \{S \to A, A \to bS \mid b\} \end{aligned}$$

Observe-se a eliminação dos símbolos C e a de V'', uma vez que eles não comparecem em qualquer forma sentencial derivável a partir de S. G'' corresponde, assim, a uma gramática equivalente a G, isenta de símbolos inacessíveis e inúteis, e $L(G) = L(G'') = b^+$.

Produções em vazio

Produções em vazio são produções da forma $A \to \varepsilon$, e a total eliminação de produções desse tipo de uma gramática G naturalmente só é possível se $\varepsilon \notin L(G)$.

Produções em vazio

Teorema 5.2 "Toda linguagem livre de contexto que não contém a cadeia vazia pode ser gerada por uma gramática livre de contexto em que não há produções em vazio."

O Algoritmo 5.3 mostra como transformar uma gramática G em outra equivalente G', isenta de produções em vazio. Se $\varepsilon \in L(G)$, então será admitida uma única produção em vazio, cujo lado esquerdo corresponde à raiz da gramática, de modo que $L(G') = L(G)$.

Produções em vazio

Algoritmo 5.3 *"Eliminação de produções em vazio em gramáticas livres de contexto."*

- *Entrada: uma gramática livre de contexto* $G = (V, \Sigma, P, S)$.
- *Saída: uma gramática livre de contexto* $G' = (N' \cup \Sigma, \Sigma, P', S')$, *tal que* $L(G') = L(G)$ *e*
  - i) *Se* $\varepsilon \notin L(G)$, *então não há produções em vazio em* G', *ou*
  - ii) *Se* $\varepsilon \in L(G)$, *então a única produção em vazio em* G' *é* $S' \to \varepsilon$, *onde* S' *é a raiz de* G' *e* S' *não aparece no lado direito de nenhuma produção.*

Produções em vazio

Método:

1. $E_0 \leftarrow \{A \mid A \rightarrow \varepsilon \in P\}$;
2. $i \leftarrow 1$;
3. $E_i \leftarrow E_{i-1} \cup \{A \mid A \rightarrow X_1 X_2 \ldots X_n \in P \text{ e } X_1, X_2 \ldots X_n \in E_{i-1}\}$;
4. *Se* $E_i \neq E_{i-1}$, *então:*
   1. $i \leftarrow i + 1$;
   2. *Desviar para (3);*

   Caso contrário:
   1. $E \leftarrow E_{i-1}$;
5. $P' \leftarrow \emptyset$;
6. $P' \leftarrow \{A \rightarrow \beta \in P \mid \beta \neq \varepsilon\}$;

Produções em vazio

7. *Considerem-se as produções de P' no formato:*
   $A \to \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \ldots B_k \alpha_k$, *com* $\alpha_i \in (V - E)^*$ *e* $B_i \in E$
   A versão final do conjunto P' é obtida acrescentando-se à sua versão anterior o conjunto das produções obtidas pela substituição dos símbolos B_i, $1 \leqslant i \leqslant k$ por ε, considerando-se todas as combinações possíveis, sem no entanto gerar a produção $A \to \varepsilon$.
8. *Se $S \in E$, então:*
   1. $P' \leftarrow P' \cup \{S' \to S \mid \varepsilon\}$;
   2. $N' \leftarrow N \cup \{S'\}$;

   Caso contrário:
   1. $N' \leftarrow N$;
   2. $S' \leftarrow S$.

Produções em vazio

Este algoritmo implementa um mapeamento do conjunto P no conjunto P', em que as seguintes condições são obedecidas:

1. Produções em vazio $A \to \varepsilon \in P$ são eliminadas;
2. As produções $A \to \alpha \in P$, $\alpha = \sigma_1\sigma_2 \ldots \sigma_n$, $\sigma_i \in V$, em que α contém símbolos não-terminais que eventualmente derivam ε, isto é, $\sigma_i \Rightarrow^* \varepsilon$, $1 \leqslant i \leqslant n$, são expandidas para contemplar todos os possíveis casos de eliminação desses não-terminais, compensando assim a eliminação das produções em vazio.

Produções em vazio

Os passos (1), (2), (3) e (4) do algoritmo buscam identificar o subconjunto E dos elementos A de N, $E \subseteq N$, tais que $A \Rightarrow^* \varepsilon$, e em muito se assemelham aos Algoritmos 5.1 e 5.2 anteriormente apresentados. Os passos (5), (6) e (7) realizam o mapeamento acima descrito, eliminando as produções em vazio e eventualmente expandindo as demais produções.

Produções em vazio

Observe-se que o fato de cada produção de P ser considerada no formato genérico:

$$A \to \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \ldots B_k \alpha_k$$

visa isolar exatamente os não-terminais B_i, tais que $B_i \Rightarrow^* \varepsilon$. A seguir são geradas, a partir de cada produção deste tipo, $2^k - 1$ (no máximo) novas produções distintas, cada qual correspondente a uma particular combinação de não-terminais B_i substituídos por ε.
Por fim, note-se que se $S \in E$, isso indica que $\varepsilon \in L(G)$. Assim, para que $L(G') = L(G)$, um novo símbolo não-terminal S' e a produção $S' \to \varepsilon$ são introduzidos em G' no passo (8).

Exemplo

Exemplo 5.2
Suponha $X \to Y\sigma Z$, com $X, Y, Z \in N$, $Y, Z \in E$ e $\sigma \in \Sigma$. Então, esta regra dá origem às seguintes novas regras:

$$\begin{aligned} X &\to \sigma Z \\ X &\to Y\sigma \\ X &\to \sigma \end{aligned}$$

Suponha agora $X \to YZ$, com $X, Y, Z \in N$ e $Y, Z \in E$. Então, esta regra dá origem às seguintes novas regras:

$$\begin{aligned} X &\to Z \\ X &\to Y \end{aligned}$$

Note que, em nenhum caso, a regra $X \to \varepsilon$ é gerada.

Exemplo

Exemplo 5.3
Seja uma gramática $G = (V, \Sigma, P, S)$, com:

$$\begin{aligned} V &= \{S, A, B, C\} \\ \Sigma &= \{a, b\} \\ P &= \{S \to ABC, A \to BB \mid \varepsilon, B \to CC \mid a, C \to AA \mid b\} \end{aligned}$$

Como se pode perceber, $E = \{A, C, B, S\}$. Em conseqüência, obtém-se o seguinte conjunto de produções P':

$$\begin{aligned} \{C &\to AA \mid A \mid b, \\ B &\to CC \mid C \mid a, \\ A &\to BB \mid B, \\ S &\to ABC \mid A \mid B \mid C \mid AB \mid AC \mid BC\} \end{aligned}$$

Pelo fato de $S \in E$, a produção $S' \to S \mid \varepsilon$ deve ainda ser incorporada a P', resultando assim $G' = (V \cup \{S'\}, \Sigma, P', S')$.

Exemplo

Exemplo 5.4
Considere-se a gramática $G = (V, \Sigma, P, S)$, com

$$\begin{aligned} V &= \{S, B, C\} \\ \Sigma &= \{a, b, c, d\} \\ P &= \{S \to aBC, B \to bB \mid \varepsilon, C \to cCc \mid d \mid \varepsilon\} \end{aligned}$$

Neste caso, $E = \{B, C\}$. Portanto, P' se torna:

$$\begin{aligned} \{B &\to bB \mid b, \\ C &\to cCc \mid cc \mid d, \\ S &\to aBC \mid aB \mid aC \mid a\} \end{aligned}$$

Como $\varepsilon \notin L(G)$, então $G' = (V, \Sigma, P', S)$.

Produções unitárias

Produções unitárias são produções da forma $A \to B$, em que A e B são não-terminais, e costumam ser descartadas das gramáticas livres de contexto porque nada acrescentam às formas sentenciais às quais são aplicadas, constituindo mera renomeação de símbolos (no caso, de A para B).

Produções unitárias

Teorema 5.3 "Toda linguagem livre de contexto pode ser gerada por uma gramática livre de contexto isenta de produções unitárias."

O Algoritmo 5.4 mostra como transformar gramáticas livres de contexto arbitrárias em outras equivalentes sem produções unitárias.

Produções unitárias

Algoritmo 5.4 *"Eliminação de produções unitárias em gramáticas livres de contexto."*

- *Entrada: uma gramática livre de contexto* $G = (V, \Sigma, P, S)$.
- *Saída: uma gramática livre de contexto* $G' = (V, \Sigma, P', S)$, *tal que* $L(G') = L(G)$ *e* G' *não contém produções unitárias.*

Produções unitárias

Método:

1. *Para cada* $A \in N$, *constrói-se* N_A *tal que* $N_A = \{B \in N \mid A \Rightarrow^* B\}$ *da seguinte forma:*
   1. $N_0 \leftarrow \{A\}$;
   2. $i \leftarrow 1$;
   3. $N_i \leftarrow N_{i-1} \cup \{C \mid B \rightarrow C \in P \text{ e } B \in N_{i-1}\}$;
   4. *Se* $N_i \neq N_{i-1}$, *então:*
      1. $i \leftarrow i+1$;
      2. *Desviar para (1.c);*

      Caso contrário:
      1. $N_A \leftarrow N_{i-1}$;
2. $P' \leftarrow \{A \rightarrow \alpha \in P \mid \alpha \notin N\}$;
3. *Para todo* $B \in N_A$, *se* $B \rightarrow \alpha \in P$, *e* $\alpha \notin N$, *então* $P' \leftarrow P' \cup \{A \rightarrow \alpha\}$.

Produções unitárias

- O funcionamento deste algoritmo baseia-se inicialmente na identificação e na associação de subconjuntos de N a cada não-terminal X da gramática, sendo que cada elemento Y desse subconjunto satisfaz à condição $X \Rightarrow^* Y$. Esses subconjuntos são construídos no passo (1) do algoritmo.
- Observe-se que a condição $X \Rightarrow^* Y$ refere-se apenas a derivações efetuadas exclusivamente através do emprego de produções unitárias. Em alguns casos, é possível que tal condição seja satisfeita através do emprego de produções não-unitárias, isto é, $X \Rightarrow^* \alpha \Rightarrow^* Y, |\alpha| > 1$, como ilustrado no Exemplo 5.5.

Exemplo

Exemplo 5.5
Considere-se a gramática G:

$$\begin{aligned} G &= (\{S,X,Y,a\},\{a\},P,S) \\ P &= \{S \to XY, \\ &\quad X \to a, \\ &\quad Y \to \varepsilon\} \end{aligned}$$

A derivação $S \Rightarrow XY \Rightarrow X$, ou simplesmente $S \Rightarrow^* X$, é feita sem o uso de regras unitárias.

Produções unitárias

Para evitar situações como as do Exemplo 5.5, é suficiente garantir que a gramática em questão não possua regras em vazio, ou, alternativamente, desconsiderar derivações que façam uso das mesmas no cálculo dos conjuntos N_A (passo (1) do Algoritmo 5.4). O conjunto P' é construído nos passos (2) e (3). Ele é obtido pela eliminação de todas as produções unitárias de P, preservando-se as demais e criando novas produções do tipo $X \Rightarrow^* \alpha$, onde $Y \in N_X$ e $Y \to \alpha \in P$. Isso implica "abreviar" uma seqüência de derivações, substituindo cada derivação que utilize produções unitárias pela derivação — utilizando as novas produções — que produz o efeito final pretendido.

Exemplo

Exemplo 5.6
Considere-se $G = (V, \Sigma, P, S)$, com o conjunto de regras P seguinte:

$$\begin{aligned} \{S &\rightarrow A \mid B \mid C, \\ A &\rightarrow aaAa \mid B \mid \varepsilon, \\ B &\rightarrow bBb \mid b \mid C, \\ C &\rightarrow cC \mid \varepsilon\} \end{aligned}$$

Aplicando-se o Algoritmo 5.4, obtém-se:

$$\begin{aligned} N_S &= \{S, A, B, C\} \\ N_A &= \{A, B, C\} \\ N_B &= \{B, C\} \\ N_C &= \{C\} \end{aligned}$$

Exemplo

Assim, $G' = (V, \Sigma, P', S)$, e P' torna-se:

$$\begin{aligned} \{S &\rightarrow aaAa \mid \varepsilon \mid bBb \mid b \mid cC, \\ A &\rightarrow aaAa \mid \varepsilon \mid bBb \mid b \mid cC, \\ B &\rightarrow bBb \mid b \mid cC \mid \varepsilon, \\ C &\rightarrow cC \mid \varepsilon\} \end{aligned}$$

A título de ilustração, considere-se a derivação da sentença *aabba*. Tomando-se como base a gramática G, a seqüência de derivações desta sentença é:

$$S \Rightarrow A \Rightarrow aaAa \Rightarrow aaBa \Rightarrow aabBba \Rightarrow aabCba \Rightarrow aabba$$

ao passo que, tomando-se como base a gramática modificada G', a seqüência de derivação desta sentença se torna:

$$S \Rightarrow aaAa \Rightarrow aabBba \Rightarrow aabba$$

(note-se como a eliminação de produções unitárias "abrevia" a derivação da sentença).

Ordem de aplicação dos algoritmos

Os algoritmos de simplificação devem ser aplicados na seguinte ordem (para que não haja necessidade de reaplicação posterior):

1. Eliminação de regras vazias (Algoritmo 5.3);
2. Eliminação de regras unitárias (Algoritmo 5.4);
3. Eliminação de símbolos inúteis (Algoritmo 5.1);
4. Eliminação de símbolos inacessíveis (Algoritmo 5.2).

Ordem de aplicação dos algoritmos

Justificativa:

1. O Algoritmo 5.3 produz uma gramática equivalente isenta de regras vazias;
2. O Algoritmo 5.4 produz uma gramática equivalente isenta de regras unitárias e não introduz regras vazias.
3. O Algoritmo 5.1 produz uma gramática equivalente isenta de símbolos inúteis e não introduz regras vazias nem regras unitárias;
4. O Algoritmo 5.2 produz uma gramática equivalente isenta de símbolos inacessíveis e não introduz símbolos inúteis nem regras vazias ou unitárias.

Detecção de ciclos

Diz-se que uma gramática livre de contexto isenta de produções em vazio é **acíclica** se e somente se não existir $A \in N$, tal que $A \Rightarrow^* A$. Caso contrário, diz-se que a gramática é **cíclica**. O Algoritmo 5.4 pode ser facilmente adaptado para detectar ciclos em gramáticas livres de contexto isentas de produções em vazio, conforme apresentado no Algoritmo 5.5.

Detecção de ciclos

Algoritmo 5.5 *"Detecção de ciclos em gramáticas livres de contexto isentas de produções em vazio."*

- *Entrada: uma gramática livre de contexto isenta de produções em vazio* $G = (V, \Sigma, P, S)$.
- *Saída: SIM, se G for cíclica. NÃO, se G for acíclica.*

Detecção de ciclos

Método:

1. *Para cada $A \in N$, constrói-se N_A tal que $N_A = \{B \in N \mid A \Rightarrow^* B\}$ da seguinte forma:*
   1. $N_0 \leftarrow \{B \mid A \rightarrow B \in P \text{ e } B \neq A\}$;
   2. $i \leftarrow 1$;
   3. $N_i \leftarrow N_{i-1} \cup \{C \mid B \rightarrow C \in P \text{ e } B \in N_{i-1}\}$;
   4. *Se $N_i \neq N_{i-1}$, então:*
      1. $i \leftarrow i+1$;
      2. *Desviar para (1.c);*

      Caso contrário:
      1. $N_A \leftarrow N_{i-1}$;
2. *Se $\forall A \in N, A \notin N_A$, então NÃO; caso contrário, SIM.*

Conceito

- Uma gramática é dita **normalizada** em relação a um certo padrão quando todas as suas produções seguem as restrições impostas pelo padrão em questão;
- É comum designar tais padrões como **formas normais**;
- Nesta seção serão definidas duas das formas normais mais importantes para as gramáticas livres de contexto: a Forma Normal de Chomsky e a Forma Normal de Greibach;
- Mostrar-se-á que toda e qualquer gramática do tipo 2 corresponde a gramáticas equivalentes, expressas em ambas as formas normais.

Forma Normal de Chomsky

Diz-se que uma gramática $G = (V, \Sigma, P, S)$ do tipo 2 obedece à **Forma Normal de Chomsky** se todas as produções $p \in P$ forem de uma das duas formas seguintes:

1. $A \to BC$, ou
2. $A \to a$

com A, B, $C \in N$ e $a \in \Sigma$.
Se $\varepsilon \in L(G)$, então admite-se $S \to \varepsilon$ como única produção em que ε comparece do lado direito.

Forma Normal de Chomsky

Teorema 6.1 "Toda linguagem livre de contexto L pode ser gerada por uma gramática livre de contexto na Forma Normal de Chomsky."

Seja G uma gramática livre de contexto qualquer que gera L. Sem perda de generalidade, considere-se que G não apresenta produções unitárias, símbolos inúteis e nem produções em vazio — exceto $S \rightarrow \varepsilon$, se $\varepsilon \in L(G)$. É possível demonstrar formalmente (ver Hopcroft79) que o mapeamento de G em G', a gramática equivalente na Forma Normal de Chomsky, pode ser efetuado através do Algoritmo 6.1.

Forma Normal de Chomsky

Algoritmo 6.1 *"Obtenção de uma gramática livre de contexto na Forma Normal de Chomsky."*

- *Entrada: uma gramática livre de contexto* $G = (V, \Sigma, P, S)$ *isenta de produções unitárias, símbolos inúteis e produções em vazio.*
- *Saída: uma gramática livre de contexto* $G' = (V', \Sigma, P', S)$*, na Forma Normal de Chomsky, tal que* $L(G) = L(G')$.

Forma Normal de Chomsky

Método:

1. $P' \leftarrow \emptyset$;
2. $N' \leftarrow N$;
3. *Se* $A \rightarrow BC \in P$, *com* $A, B, C \in N$, *então* $A \rightarrow BC \in P'$;
4. *Se* $A \rightarrow \sigma \in P$, *com* $A \in N, \sigma \in \Sigma$, *então* $A \rightarrow \sigma \in P'$;
5. *Se* $S \rightarrow \varepsilon \in P$, *então* $S \rightarrow \varepsilon \in P'$;
6. *Para cada produção* $p \in P$ *da forma:*
   $A \rightarrow X_1 X_2, \ldots, X_n$, *com* $n \geqslant 2$
   se $X_i \in \Sigma$, *então criam-se novos não-terminais* Y_i *e produções* $Y_i \rightarrow X_i$ *substituindo-se as ocorrências de* X_i *por* Y_i *em p. Acrescentam-se os novos não-terminais* Y_i *a* N' *e as novas produções a* P'.

Forma Normal de Chomsky

7. *Para cada produção da forma:*
   $A \rightarrow X_1 X_2, \ldots, X_n$, *com* $n > 2$ *e* $X_i \in N$, $1 \leqslant i \leqslant n$
   gerada no passo (6), criar um novo conjunto de não-terminais Z_i *e de produções da forma:*

$$\begin{aligned} \{A &\rightarrow X_1 Z_1, \\ Z_1 &\rightarrow X_2 Z_2, \\ &\ldots \\ Z_{n-2} &\rightarrow X_{n-1} X_n\} \end{aligned}$$

   acrescentando-os, respectivamente, aos conjuntos N' *e* P'.

Exemplo

Exemplo 6.1
Considere-se a gramática $G = (V, \Sigma, P, S)$ com $N = \{E, T, F\}$ e o conjunto de produções P apresentado abaixo:

$$\begin{aligned} \{E &\rightarrow E + T \mid T, \\ T &\rightarrow T * F \mid F, \\ F &\rightarrow (E) \mid a\} \end{aligned}$$

Exemplo

Da aplicação do algoritmo acima resulta $G' = (V', \Sigma, P', S)$, com:

$$
\begin{aligned}
N' &= \{E, T, F, X_0, X_1, X_2, X_3, W_0, W_1, W_2\} \\
P' &= \{E \to EW_0 | TW_1 | X_2W_2 | a, \\
&\quad W_0 \to X_0T, \\
&\quad W_1 \to X_1F, \\
&\quad W_2 \to EX_3, \\
&\quad T \to TW_1 | X_2W_2 | a, \\
&\quad F \to X_2W_2 | a, \\
&\quad X_0 \to +, \\
&\quad X_1 \to *, \\
&\quad X_2 \to (, \\
&\quad X_3 \to)\}"
\end{aligned}
$$

Forma Normal de Greibach

Diz-se que uma gramática livre de contexto $G = (V, \Sigma, P, S)$ obedece à **Forma Normal de Greibach** se todas as suas produções $p \in P$ forem da forma:

$$A \to \sigma\alpha, \sigma \in \Sigma, \alpha \in N^*$$

Se $\varepsilon \in L(G)$, então admite-se $S \to \varepsilon$ como única produção em que ε comparece do lado direito. Como pré-requisito antes de apresentarmos o teorema que mostra como converter gramáticas livres de contexto quaisquer em equivalentes na Forma Normal de Greibach, será necessário apresentar um algoritmo que permita eliminar recursões à esquerda em gramáticas livres de contexto quaisquer.

Forma Normal de Greibach

Eliminar recursões à esquerda de uma gramática livre de contexto $G = (V, \Sigma, P, S)$ significa obter $G' = (V', \Sigma, P', S)$, de modo que $L(G) = L(G')$ e nenhum $A \in N'$ seja recursivo à esquerda. A eliminação de recursões à esquerda em gramáticas livres de contexto pode ser feita através de um algoritmo cuja idéia básica consiste em transformar G, com não-terminais ordenados arbitrariamente, em G', de modo que $X_i \rightarrow X_j\alpha$ pertence a P' se e apenas se $j > i$.
Sem perda de generalidade, admite-se que a gramática G de entrada seja isenta de produções unitárias, símbolos inúteis e produções em vazio.

Forma Normal de Greibach

Teorema 6.2 "Toda linguagem livre de contexto L pode ser gerada por uma gramática livre de contexto na Forma Normal de Greibach."

Pode ser desenvolvida a partir do Algoritmo 6.2, que permite a conversão de uma gramática livre de contexto qualquer em outra equivalente na Forma Normal de Greibach.

Forma Normal de Greibach

Algoritmo 6.2 *"Obtenção de uma gramática livre de contexto na Forma Normal de Greibach."*

- *Entrada: uma gramática livre de contexto* $G = (V, \Sigma, P, S)$ *isenta de produções unitárias, símbolos inacessíveis, símbolos inúteis, produções em vazio e recursões à esquerda, tal que* $L = L(G)$.
- *Saída: uma gramática livre de contexto* $G' = (V', \Sigma, P', S)$ *na Forma Normal de Greibach, tal que* $L(G) = L(G')$.

Conceito

- Os **autômatos de pilha** constituem a segunda instância, em uma escala de complexidade crescente, do modelo genérico de reconhecedor introduzido anteriormente, prestando-se ao reconhecimento de linguagens livres de contexto;
- Diferentemente dos autômatos finitos, que não se utilizam da memória auxiliar prevista no modelo genérico, os autômatos de pilha têm o seu poder de reconhecimento estendido, quando comparado ao dos autômatos finitos, justamente pela disponibilidade e pela utilização de uma memória auxiliar organizada na forma de uma pilha.

Pilha

A pilha de um autômato de pilha pode ser entendida como uma estrutura de dados, de capacidade ilimitada, na qual a máquina de estados é capaz de armazenar, consultar e remover símbolos de um alfabeto próprio, denominado **alfabeto de pilha**, segundo a convenção usual para estruturas deste tipo (LIFO — “last-in-first-out”). Quanto aos seus demais componentes e características, o autômato de pilha se assemelha ao autômato finito anteriormente estudado.

Definição

Formalmente, um autômato de pilha pode ser definido como uma sétupla M:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

onde:

Q	é um conjunto finito de estados;
Σ	é um alfabeto (finito e não-vazio) de entrada;
Γ	é um alfabeto (finito e não-vazio) de pilha;
δ	é uma função de transição $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$;
q_0	é o estado inicial de M, $q_0 \in Q$;
Z_0	é o símbolo inicial da pilha, $Z_0 \in \Gamma$;
F	é o conjunto de estados finais de M, $F \subseteq Q$.

Dos sete elementos que compõem este sistema formal, apenas três merecem explicação adicional — δ, Γ e Z_0 —, uma vez que os demais estão em correspondência direta com o que já foi estudado para o caso dos autômatos finitos.

Símbolo inicial da pilha

Note-se, inicialmente, a presença de um alfabeto de pilha Γ e de um símbolo inicial de pilha Z_0. O alfabeto de pilha especifica os símbolos que podem ser armazenados na pilha. Por convenção, um desses símbolos, denotado Z_0, representa o conteúdo inicial da pilha toda vez que o autômato de pilha principia o reconhecimento de uma nova sentença. Ao longo de sua operação, elementos de Γ são acrescentados e/ou removidos da pilha, e seu conteúdo total pode ser interpretado, em um dado instante, como sendo um elemento de Γ^*. Por convenção, cadeias de Γ que representam o conteúdo da pilha em um determinado instante são interpretadas considerando-se os símbolos mais à esquerda da cadeia no topo da pilha, e os símbolos mais à direita da cadeia no fundo da pilha.

Função de transição

Observe-se que, a partir do formato geral apresentado acima para a função de transição δ em autômatos de pilha não-determinísticos e com transições em vazio, é possível extrair os seguintes casos particulares, equivalentes aos estudados do Capítulo 3, para os autômatos finitos:

- Autômato de pilha determinístico sem transições em vazio: $\delta : Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$
- Autômato de pilha determinístico com transições em vazio: $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$
- Autômato de pilha não-determinístico sem transições em vazio: $Q \times \Sigma \times \Gamma \to 2^{Q \times \Gamma^*}$
- Autômato de pilha não-determinístico com transições em vazio: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$

Exemplo

Exemplo 7.1
Considere-se $\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$. A cadeia $\gamma_2\gamma_1\gamma_4\gamma_4$, por exemplo, representa o conteúdo da pilha conforme ilustrado na Figura 5. Note-se que γ_2 se encontra no topo da pilha e que γ_4 se encontra no fundo da mesma:

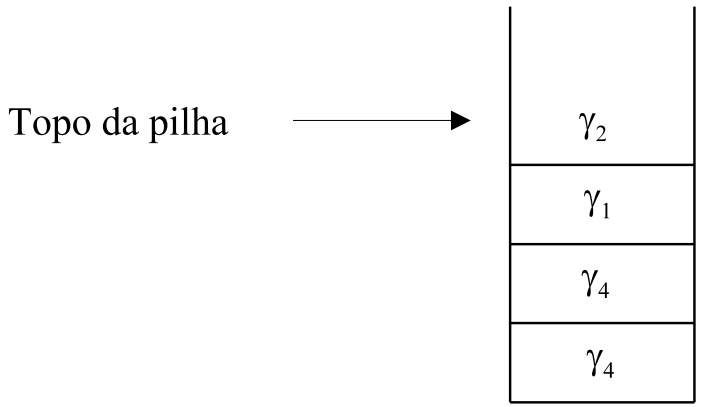

Figura 5: Pilha com ponteiro de topo

“Stack” × “pushdown”

Os dispositivos que fazem uso de pilha costumam definir a forma de operação da mesma como uma das seguintes possibilidades:

- **Pilha “Stack”**: além das operações de empilhamento e desempilhamento de elementos no topo da pilha (“push” e “pop”), permite que os demais elementos da mesma sejam endereçados diretamente, somente para consulta;
- **Pilha “Pushdown”**: permite o acesso apenas ao elemento armazenado no topo da pilha, através das operações de empilhamento e desempilhamento (“push” e “pop”). Não permite o endereçamento dos demais elementos da pilha.

A definição de autômato de pilha neste texto considera que a sua memória auxiliar seja uma pilha “pushdown”.

Configuração e configuração inicial

A **configuração de um autômato de pilha** é definida pelo seu estado corrente, pela parte da cadeia de entrada ainda não analisada e pelo conteúdo da pilha. A **configuração inicial de um autômato de pilha** é aquela em que o autômato se encontra no estado inicial q_0, o cursor se encontra posicionado sob a célula mais à esquerda na fita de entrada e o conteúdo da pilha é Z_0. Algebricamente, a configuração de um autômato de pilha pode ser representada como uma tripla $(q, \alpha, \gamma) \in Q \times \Sigma^* \times \Gamma^*$.
A configuração inicial para o reconhecimento de uma cadeia w é representada como (q_0, w, Z_0).

Movimentação

- As possibilidades de movimentação de um autômato de pilha em uma dada configuração são determinadas a partir de três informações: o seu estado corrente, o próximo símbolo presente na cadeia de entrada e o símbolo armazenado no topo da pilha;
- Observe-se, pela definição da função δ, a possibilidade de movimentações em vazio, sem consumo de símbolos da fita de entrada, e também a possibilidade de serem especificadas transições não-determinísticas;
- Note-se também a obrigatoriedade, imposta por essa formulação, de se consultar o símbolo presente no topo da pilha em toda e qualquer transição efetuada pelo autômato.

Movimentação

- Após a aplicação de uma transição, o cursor de leitura sofre um deslocamento de uma posição para a direita, e o símbolo presente no topo da pilha é removido, sendo substituído pela cadeia de símbolos especificada no lado direito da transição;
- No caso de transições em vazio, em que não há consulta de símbolo na fita de entrada, a posição do cursor permanece inalterada após a sua aplicação.

Movimentação

- Desejando-se efetuar alguma transição de forma independente do conteúdo do topo da pilha, torna-se necessário especificar para cada elemento do alfabeto de pilha uma transição que efetue o mesmo tratamento do símbolo de entrada, removendo e reinserindo o mesmo símbolo da pilha, simulando assim uma transição independente do conteúdo da pilha.

Autômato determinístico

Para cada tripla (q, σ, γ) pertencente ao domínio da função δ, o elemento $\delta(q, \sigma, \gamma)$ pode conter zero, um ou mais elementos de $Q \times \Gamma^*$. Considere-se inicialmente o caso em que $\sigma \neq \varepsilon$. Havendo zero elementos em $\delta(q, \sigma, \gamma)$, isso indica que não há possibilidade de movimentação a partir da configuração considerada. Havendo um único elemento, isso significa que há apenas uma possibilidade de movimentação e, portanto, a transição é determinística. Quando todas as transições de um autômato de pilha são determinísticas, diz-se que o mesmo é **determinístico**.

Autômato não-determinístico

Nos casos em que há mais de um elemento em $\delta(q, \sigma, \gamma)$, isso significa que há mais de uma opção de movimentação a partir desta configuração, e então a transição é dita não-determinística. Havendo pelo menos uma transição não-determinística, diz-se que o autômato de pilha é **não-determinístico**.

Autômato não-determinístico

Autômatos de pilha não-determinísticos e com transições em vazio apresentam um comportamento dinâmico determinístico quando as duas seguintes condições forem simultaneamente verificadas:

1. $\forall q \in Q$, $\gamma \in \Gamma$, se $|\delta(q, \varepsilon, \gamma)| \geqslant 1$, então $|\delta(q, \sigma, \gamma)| = 0$, $\forall \sigma \in \Sigma$;
2. $\forall q \in Q$, $\gamma \in \Gamma$, $\sigma \in \Sigma \cup \{\varepsilon\}, |\delta(q, \sigma, \gamma))| \leqslant 1$.

Determinismo $\times$ não-determinismo

- Diferentemente do que foi visto para o caso dos autômatos finitos, no que se refere à equivalência dos modelos determinístico e não-determinístico quanto à classe de linguagens que são capazes de reconhecer, os autômatos de pilha não apresentam correspondente equivalência;
- Conforme será visto mais adiante, os autômatos de pilha determinísticos são capazes de reconhecer apenas um subconjunto das linguagens livres de contexto;
- Por esse motivo, exceto por ressalvas em sentido contrário, os autômatos de pilha mencionados daqui em diante serão não-determinísticos.

Movimentação

A movimentação de um autômato de pilha de uma configuração para a configuração seguinte é denotada pelo símbolo "$\vdash$", que representa a relação:

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

Estando o autômato de pilha em uma configuração $(q_i, \sigma\alpha, \phi\gamma)$, com $q_i \in Q$, $\sigma \in \Sigma$, $\alpha \in \Sigma^*$, $\phi \in \Gamma$ e $\gamma \in \Gamma^*$, sua movimentação a partir dessa configuração para a seguinte, como decorrência da aplicação de uma transição $\delta(q_i, \sigma, \phi) = (q_j, \eta)$, com $\eta \in \Gamma^*$, é representada por:

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

Movimentação

A aplicação de transições em vazio, em que não há consumo de símbolo, é representada de maneira similar através de:

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

Da mesma forma que no caso dos autômatos finitos, a aplicação de zero ou mais transições entre duas configurações quaisquer é representada através do símbolo $\vdash^*$, e a aplicação de no mínimo uma transição, por intermédio do símbolo $\vdash^+$.

Configuração final

A **configuração final de um autômato de pilha** costuma ser caracterizada de duas maneiras distintas, porém equivalentes. Na primeira delas, exige-se o esgotamento da cadeia de entrada e também que o autômato atinja um estado final. Nesta caracterização, o conteúdo final da pilha é irrelevante. Na segunda caracterização, exige-se o esgotamento da cadeia de entrada e também que a pilha tenha sido completamente esvaziada, não importando que o estado atingido seja final ou não-final.
Dependendo do tipo de definição adotada para caracterizar a configuração final de um autômato de pilha, é possível definir, também de duas maneiras distintas, a linguagem aceita pelo dispositivo.

Critério de aceitação “estado final”

A linguagem aceita por um autômato de pilha M, com base no **critério de estado final**, denotada $L(M)$, é definida como:

$$L(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma), q \in F, \gamma \in \Gamma^*\}$$

Critério de aceitação “pilha vazia”

Analogamente, a linguagem aceita por um autômato de pilha M, com base no **critério de pilha vazia**, denotada $V(M)$, é definida como:

$$V(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon), q \in Q\}$$

Notação

Com o intuito de facilitar a leitura do texto no restante deste capítulo, as transições de um autômato de pilha serão denotadas como:

$$(q_i, \sigma, X) \rightarrow (q_j, \gamma)$$

indicando, com isso, o par ordenado $((q_i, \sigma, X), (q_j, \gamma))$ pertencente à função δ, ou, simplesmente, $\delta(q_i, \sigma, X) = (q_j, \gamma)$.

Diagramas de estado

Convém, neste ponto, introduzir uma extensão na notação dos Diagramas de Estado estudados para o caso dos autômatos finitos, com o intuito de permitir a representação gráfica também dos autômatos de pilha. Diferentemente dos Diagramas de Estado definidos para os autômatos finitos, os arcos entre dois estados p e q são rotulados com cadeias da forma:

$$(\sigma, Z)/\gamma, \text{ com } \sigma \in \Sigma, Z \in \Gamma \text{ e } \gamma \in \Gamma^*$$

para cada produção $\delta(p, \sigma, Z) = (q, \gamma)$.

Exemplo 1

Exemplo 7.2
Seja M_0 o autômato de pilha determinístico apresentado abaixo e ilustrado por meio do Diagrama de Estados da Figura 6, cujo critério de aceitação de sentenças é baseado no esvaziamento da pilha.

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{Z_0, X\} \\
\delta &= \{(q_0, a, Z_0) \rightarrow \{(q_0, X)\}, \\
&\quad (q_0, a, X) \rightarrow \{(q_0, XX)\}, \\
&\quad (q_0, b, X) \rightarrow \{(q_1, \varepsilon)\}, \\
&\quad (q_1, b, X) \rightarrow \{(q_1, \varepsilon)\}, \\
&\quad (q_0, \varepsilon, Z_0) \rightarrow \{(q_0, \varepsilon)\}\} \\
F &= \emptyset
\end{aligned}
$$

Exemplo 1

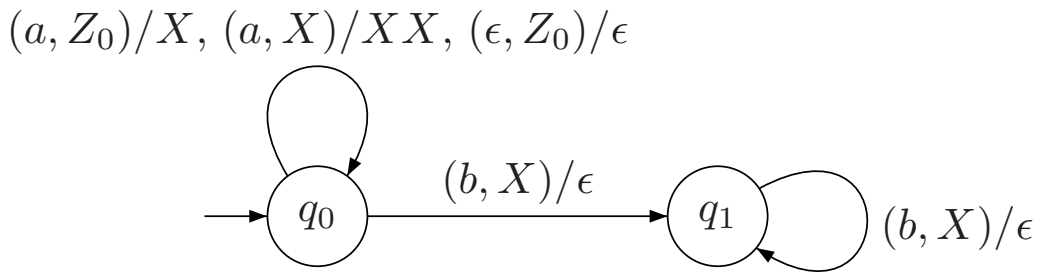

Figura 6: Autômato de pilha M_0 para $a^n b^n$ com $n \geq 0$

- Este autômato aceita a linguagem $a^k b^k$, com $k \geq 0$, que é não-regular e livre de contexto;
- M_0 opera empilhando tantos X quantos sejam os as que ele encontra na cadeia de entrada. Em seguida, ele desempilha um X para cada b encontrado na cadeia de entrada. Dessa forma, o critério de aceitação por pilha vazia garante que os bs ocorrem depois dos as e, além disso, que a quantidade de as é igual à quantidade de bs.

Exemplo 1

- Cadeias aceitas por M_0:
  - ε: $(q_0, \varepsilon, Z_0) \vdash (q_0, \varepsilon, \varepsilon)$
  - ab: $(q_0, ab, Z_0) \vdash (q_0, b, X) \vdash (q_1, \varepsilon, \varepsilon)$
  - $aabb$: $(q_0, aabb, Z_0) \vdash (q_0, abb, X) \vdash (q_0, bb, XX) \vdash (q_1, b, X) \vdash (q_1, \varepsilon, \varepsilon)$
- Cadeias rejeitadas por M_0:
  - ba: (q_0, ba, Z_0)
  - abb: $(q_0, abb, Z_0) \vdash (q_0, bb, X) \vdash (q_1, b, \varepsilon)$
  - aba: $(q_0, aba, Z_0) \vdash (q_0, ba, X) \vdash (q_1, a, \varepsilon)$
  - aab: $(q_0, aab, Z_0) \vdash (q_0, ab, X) \vdash (q_0, b, XX) \vdash (q_1, \varepsilon, X)$

Exemplo 2

Exemplo 7.3
Seja M_1 o autômato de pilha determinístico abaixo discriminado, cujo critério de aceitação de sentenças é baseado no esvaziamento da pilha.

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b, c\} \\
\Gamma &= \{Z_0, C\} \\
\delta &= \{(q_0, a, Z_0) \rightarrow \{(q_0, CCZ_0)\}, \\
&\quad (q_0, a, C) \rightarrow \{(q_0, CCC)\}, \\
&\quad (q_0, b, Z_0) \rightarrow \{(q_1, Z_0)\}, \\
&\quad (q_0, b, C) \rightarrow \{(q_1, C)\}, \\
&\quad (q_1, c, C) \rightarrow \{(q_1, \varepsilon)\}, \\
&\quad (q_1, \varepsilon, Z_0) \rightarrow \{(q_1, \varepsilon)\}\} \\
F &= \emptyset
\end{aligned}
$$

Exemplo 2

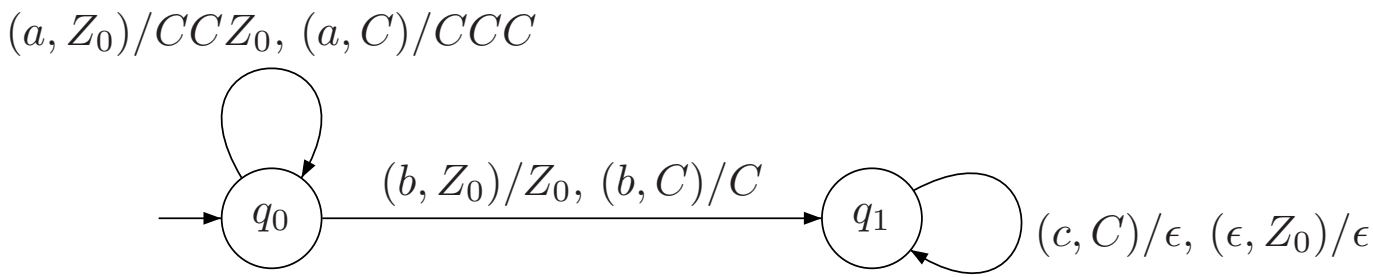

Figura 7: Autômato de pilha M_1 para $a^i bc^{2i}$, com $i \geqslant 0$

Exemplo 2

Deve-se notar, em primeiro lugar, que o determinismo deste autômato decorre do fato de que a função δ exibe no máximo um elemento para cada combinação possível de estado, símbolo de entrada e símbolo no topo da pilha, e também porque inexistem transições em vazio que se refiram a pares de estado e símbolo no topo da pilha considerados nas transições não-vazias: note-se que a única transição em vazio deste autômato ocorre no estado q_1, com um símbolo do alfabeto de pilha (Z_0) diferente do utilizado na transição não-vazia desse mesmo estado (C).
A linguagem definida por M_1 é:

$$V_1(M_1) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon), q \in Q\}$$

Como conseqüência, o conjunto de sentenças capazes de conduzir este autômato a configurações finais, com base no critério de pilha vazia, e portanto a linguagem por ele definida, é o seguinte:

$$V_1(M_1) = \{a^i b c^{2i} \mid i \geqslant 0\}$$

Exemplo 2

Considerem-se algumas sentenças desta linguagem e a correspondente seqüência de movimentos executada pelo autômato durante o seu reconhecimento:

1. Sentença: b

   Movimentos: $(q_0, b, Z_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

2. Sentença: $abcc$

   Movimentos: $(q_0, abcc, Z_0) \vdash (q_0, bcc, CCZ_0) \vdash (q_1, cc, CCZ_0) \vdash (q_1, c, CZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

3. Sentença: $aabcccc$

   Movimentos: $(q_0, aabcccc, Z_0) \vdash (q_0, abcccc, CCZ_0) \vdash (q_0, bcccc, CCCCZ_0) \vdash (q_1, cccc, CCCCZ_0) \vdash (q_1, ccc, CCCZ_0) \vdash (q_1, cc, CCZ_0) \vdash (q_1, c, CZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

Exemplo 2

Tome-se agora como exemplo duas sentenças que não pertencem à linguagem definida por este autômato, e as correspondentes seqüências de configurações:

1. Sentença: *abccc*

   Movimentos:
   $(q_0, abccc, Z_0) \vdash (q_0, bccc, CCZ_0) \vdash (q_1, ccc, CCZ_0) \vdash (q_1, cc, CZ_0) \vdash (q_1, c, Z_0)$

2. Sentença: *aabccc*

   Movimentos: $(q_0, aabccc, Z_0) \vdash (q_0, abccc, CCZ_0) \vdash (q_0, bccc, CCCCZ_0) \vdash (q_1, ccc, CCCCZ_0) \vdash (q_1, cc, CCCZ_0) \vdash (q_1, c, CCZ_0) \vdash (q_1, \varepsilon, CZ_0)$

Exemplo 3

Exemplo 7.4
O autômato M_2, ilustrado na Figura 8, com critério de aceitação baseado no esvaziamento da pilha, reconhece a linguagem:

$$V_2(M_2) = \{a^{2i}bc^i \mid i \geqslant 0\}$$

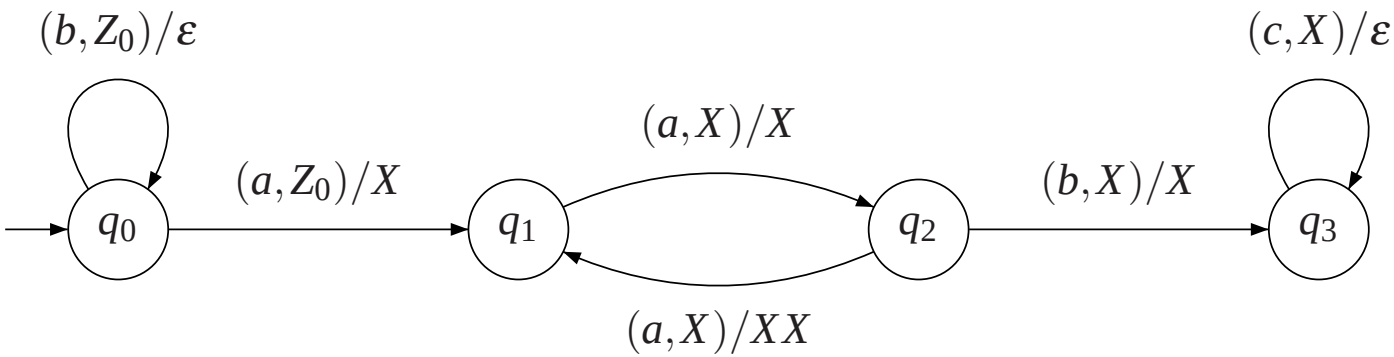

Figura 8: Autômato de pilha do Exemplo 7.4

Exemplo 4

Exemplo 7.5

Considere-se o autômato de pilha não-determinístico M_3 a seguir apresentado, cujo critério de aceitação de sentenças é baseado em estado final.

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2, q_3, q_4\} \\
\Sigma &= \{a, c\} \\
\Gamma &= \{Z_0, C\} \\
\delta &= \{(q_0, a, Z_0) \rightarrow \{(q_1, CCZ_0), (q_2, CZ_0)\}, \\
&\quad (q_1, a, C) \rightarrow \{(q_1, CCC)\}, \\
&\quad (q_1, c, C) \rightarrow \{(q_3, \varepsilon)\}, \\
&\quad (q_2, a, C) \rightarrow \{(q_2, CC)\}, \\
&\quad (q_2, c, C) \rightarrow \{(q_3, \varepsilon)\}, \\
&\quad (q_3, c, C) \rightarrow \{(q_3, \varepsilon)\}, \\
&\quad (q_3, \varepsilon, Z_0) \rightarrow \{(q_4, Z_0)\}\} \\
F &= \{q_4\}
\end{aligned}
$$

Exemplo 4

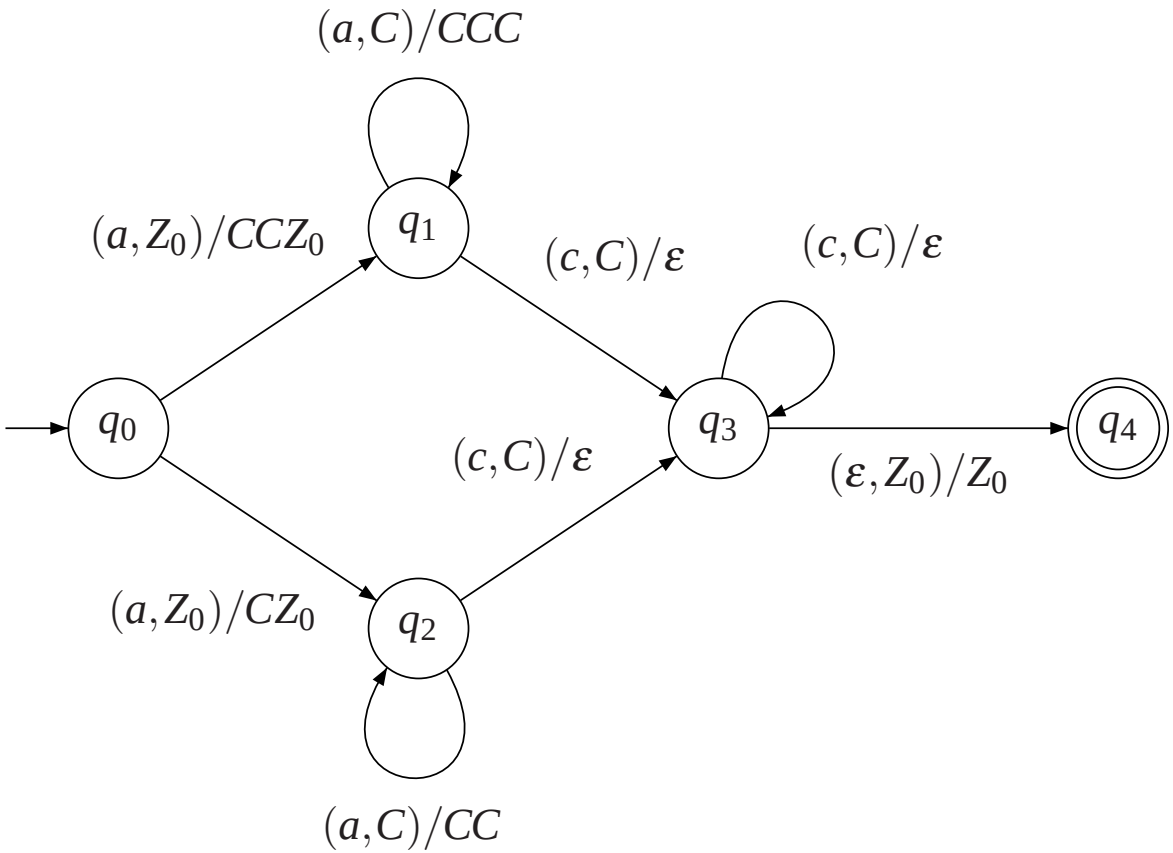

Figura 9: Autômato de pilha do Exemplo 7.5

$$L_3(M_3) = \{a^i c^j \mid i \geqslant 1 \text{ e } (j = i \text{ ou } j = 2i)\}$$

Exemplo 4

A operação deste autômato é exemplificada a seguir através do reconhecimento das seguintes sentenças:

1. Sentença: *aacc*

   Movimentos: $(q_0, aacc, Z_0) \vdash (q_2, acc, CZ_0) \vdash (q_2, cc, CCZ_0) \vdash (q_3, c, CZ_0) \vdash (q_3, \varepsilon, Z_0) \vdash (q_4, \varepsilon, Z_0)$

2. Sentença: *aacccc*

   Movimentos: $(q_0, aacccc, Z_0) \vdash (q_1, acccc, CCZ_0) \vdash (q_1, cccc, CCCCZ_0) \vdash (q_3, ccc, CCCZ_0) \vdash (q_3, cc, CCZ_0) \vdash (q_3, c, CZ_0) \vdash (q_3, \varepsilon, Z_0) \vdash (q_4, \varepsilon, Z_0)$

Note-se que o reconhecimento, em ambos os casos, pôde ser bem-sucedido já na primeira seqüência de movimentos, uma vez que a escolha da transição a ser aplicada na configuração inicial foi corretamente "adivinhada" nas duas situações.

Exemplo 4

No entanto, a escolha da transição a ser aplicada na configuração inicial do primeiro caso poderia ter sido diferente da apresentada, e neste caso ocorreria o seguinte:

1. Sentença: *aacc*

   Movimentos: $(q_0, aacc, Z_0) \vdash (q1, acc, CCZ_0) \vdash (q_1, cc, CCCCZ_0) \vdash (q_3, c, CCCZ_0) \vdash (q_3, \varepsilon, CCZ_0)$

Exemplo 4

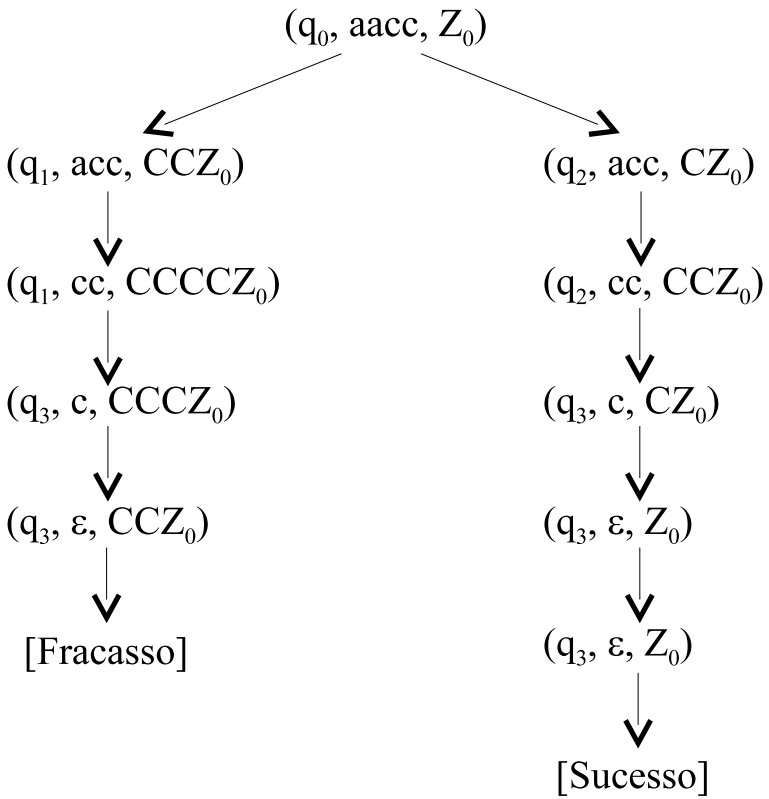

Figura 10: Aceitação em autômato de pilha não-determinístico do Exemplo 7.5

Exemplo 4

Considere-se, agora, o caso de uma cadeia — *aac* — que não pertença à linguagem definida por este autômato. Conforme ilustrado na Figura 11, a rejeição desta cadeia ocorre apenas após o fracasso do reconhecimento em todas as seqüências possíveis de movimentação (duas, para este autômato):

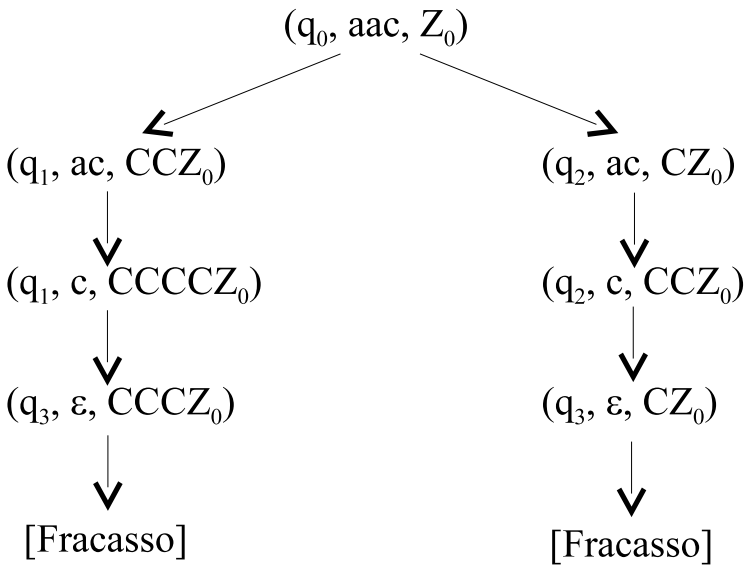

Figura 11: Rejeição em autômato de pilha não-determinístico do Exemplo 7.5

Exercícios

Obter autômatos de pilha (determinísticos ou não-determinísticos) que aceitem cada uma das seguintes linguagens:

1. $\{a^i b^i | i \geqslant 0\}$, com aceitação por estado final;
2. $\{a^i b^i | i \geqslant 0\}$, com aceitação por pilha vazia;
3. $\{a^i b^i | i \geqslant 1\}$, com aceitação por estado final;
4. $\{a^i b^i | i \geqslant 1\}$, com aceitação por pilha vazia;
5. $\{a^* b^*\}$, com aceitação por estado final;
6. $\{a^* b^*\}$, com aceitação por pilha vazia;
7. $\{a^i b^* c^i | i \geqslant 0\}$, com aceitação por estado final;
8. $\{a^i b^* c^i | i \geqslant 0\}$, com aceitação por pilha vazia;
9. $\{a^i b^* c^i | i \geqslant 1\}$, com aceitação por estado final;
10. $\{a^i b^* c^i | i \geqslant 1\}$, com aceitação por pilha vazia.

Transições em vazio

Para finalizar, cumpre notar que, de acordo com a definição, os autômatos de pilha são capazes de efetuar movimentos independentemente da existência de símbolos na fita de entrada, através das chamadas transições em vazio, ao passo que o esvaziamento da pilha necessariamente impede qualquer possibilidade de movimentação futura. Ambos estes fatos serão utilizados em seguida para demonstrar a equivalência dos critérios de aceitação por estado final e por pilha vazia.

Equivalência dos critérios de aceitação

A classe de linguagens aceita por autômatos de pilha não-determinísticos com critério de aceitação baseado em estado final é idêntica à classe de linguagens aceita por autômatos de pilha não-determinísticos com critério de aceitação baseado em pilha vazia. A importância desse resultado deve-se à liberdade de escolha que ele oferece quando se pretende demonstrar algum teorema relativo aos autômatos de pilha e às linguagens livres de contexto, podendo-se optar indistintamente entre um e outro critério de aceitação, sem prejuízo para a sua generalização.

Estado final $\Rightarrow$ pilha vazia

Teorema 8.1 "Seja M um autômato de pilha não-determinístico com critério de aceitação baseado em estado final. Então, é possível definir um autômato de pilha não-determinístico M' com critério de aceitação baseado em pilha vazia, de modo que $L(M) = V(M')$."

O Algoritmo 8.1 apresenta um método para se efetuar tal conversão.

Estado final $\Rightarrow$ pilha vazia

Algoritmo 8.1 *"Obtenção de um autômato de pilha baseado em critério de aceitação de pilha vazia a partir de outro baseado em critério de estado final."*

- *Entrada: um autômato de pilha não-determinístico* $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$*, com critério de aceitação baseado em estado final.*
- *Saída: um autômato de pilha não-determinístico* $M' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', \emptyset)$*, com critério de aceitação baseado em pilha vazia.*

Estado final $\Rightarrow$ pilha vazia

Método:

1. $Q' \leftarrow Q \cup \{q_v, q'_0\}$ *(supondo-se, sem perda de generalidade, que* $Q \cap \{q_v, q'_0\} = \emptyset$*);*
2. $\Gamma' \leftarrow \Gamma \cup \{Z'_0\}$ *(supondo-se, sem perda de generalidade, que* $\Gamma \cap \{Z'_0\} = \emptyset$*);*
3. *Função de transição* δ'*:*
   1. $\delta' \leftarrow \emptyset$;
   2. $\delta'(q'_0, \varepsilon, Z'_0) \leftarrow \{(q_0, Z_0 Z'_0)\}$;
   3. $\delta'(q, \sigma, \phi) \leftarrow \delta(q, \sigma, \phi), \quad \forall q \in Q, \phi \in \Gamma, \sigma \in (\Sigma \cup \{\varepsilon\})$;
   4. $\delta'(q, \varepsilon, \phi) \leftarrow \delta'(q, \varepsilon, \phi) \cup \{(q_v, \varepsilon)\}, \quad \forall q \in F, \phi \in (\Gamma \cup \{Z'_0\})$;
   5. $\delta'(q_v, \varepsilon, \phi) \leftarrow \{(q_v, \varepsilon)\}, \quad \forall \phi \in (\Gamma \cup \{Z'_0\})$.

Exemplo

Exemplo 8.1
Considere-se o autômato de pilha M:

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{a, b, c\} \\
\Gamma &= \{Z_0, A, B\} \\
\delta &= \{(q_0, a, Z_0) \to \{(q_0, AZ_0)\}, (q_1, a, A) \to \{(q_1, \varepsilon)\}, \\
&\quad (q_0, a, A) \to \{(q_0, AA)\}, (q_1, b, B) \to \{(q_1, \varepsilon)\}, \\
&\quad (q_0, a, B) \to \{(q_0, AB)\}, (q_1, \varepsilon, Z_0) \to \{(q_2, \varepsilon)\}, \\
&\quad (q_0, b, Z_0) \to \{(q_0, BZ_0)\}, (q_0, b, A) \to \{(q_0, BA)\}, \\
&\quad (q_0, b, B) \to \{(q_0, BB)\}, (q_0, c, Z_0) \to \{(q_2, \varepsilon)\}, \\
&\quad (q_0, c, A) \to \{(q_1, A)\}, (q_0, c, B) \to \{(q_1, B)\}\} \\
F &= \{q_2\}
\end{aligned}
$$

Exemplo

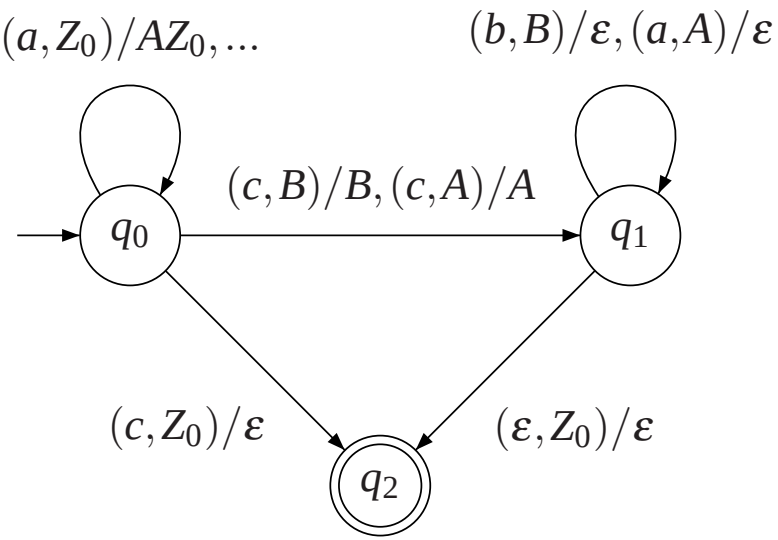

Figura 12: Autômato M com critério de aceitação de estado final

A Figura 12 ilustra M na notação dos Diagramas de Estados. A linguagem definida por ele é $\{\alpha c \alpha^R \mid \alpha \in \{a, b\}^*\}$.

Exemplo

$$
\begin{aligned}
M' &= (Q', \Sigma, \Gamma', \delta', q_0', Z_0', \emptyset) \\
Q' &= \{q_0, q_1, q_2, q_v, q_0'\} \\
\Gamma' &= \{Z_0, A, B, Z_0'\} \\
\delta' &= \{(q_0, a, Z_0) \to \{(q_0, AZ_0)\}, (q_1, a, A) \to \{(q_1, \varepsilon)\}, \\
&\quad (q_0, a, A) \to \{(q_0, AA)\}, (q_1, b, B) \to \{(q_1, \varepsilon)\}, \\
&\quad (q_0, a, B) \to \{(q_0, AB)\}, (q_1, \varepsilon, Z_0) \to \{(q_2, \varepsilon)\}, \\
&\quad (q_0, b, Z_0) \to \{(q_0, BZ_0)\}, (q_0, b, A) \to \{(q_0, BA)\}, \\
&\quad (q_0, b, B) \to \{(q_0, BB)\}, (q_0, c, Z_0) \to \{(q_2, \varepsilon)\}, \\
&\quad (q_0, c, A) \to \{(q_1, A)\}, (q_0, c, B) \to \{(q_1, B)\}, \\
&\quad (q_0', \varepsilon, Z_0') \to \{(q_0, Z_0 Z_0')\}, \\
&\quad (q_2, \varepsilon, Z_0') \to \{(q_v, \varepsilon)\}, \\
&\quad (q_2, \varepsilon, Z_0) \to \{(q_v, \varepsilon)\}, \\
&\quad (q_2, \varepsilon, A) \to \{(q_v, \varepsilon)\}, \\
&\quad (q_2, \varepsilon, B) \to \{(q_v, \varepsilon)\}, \\
&\quad (q_v, \varepsilon, Z_0') \to \{(q_v, \varepsilon)\}, \\
&\quad (q_v, \varepsilon, Z_0) \to \{(q_v, \varepsilon)\}, \\
&\quad (q_v, \varepsilon, A) \to \{(q_v, \varepsilon)\}, \\
&\quad (q_v, \varepsilon, B) \to \{(q_v, \varepsilon)\}\}
\end{aligned}
$$

Exemplo

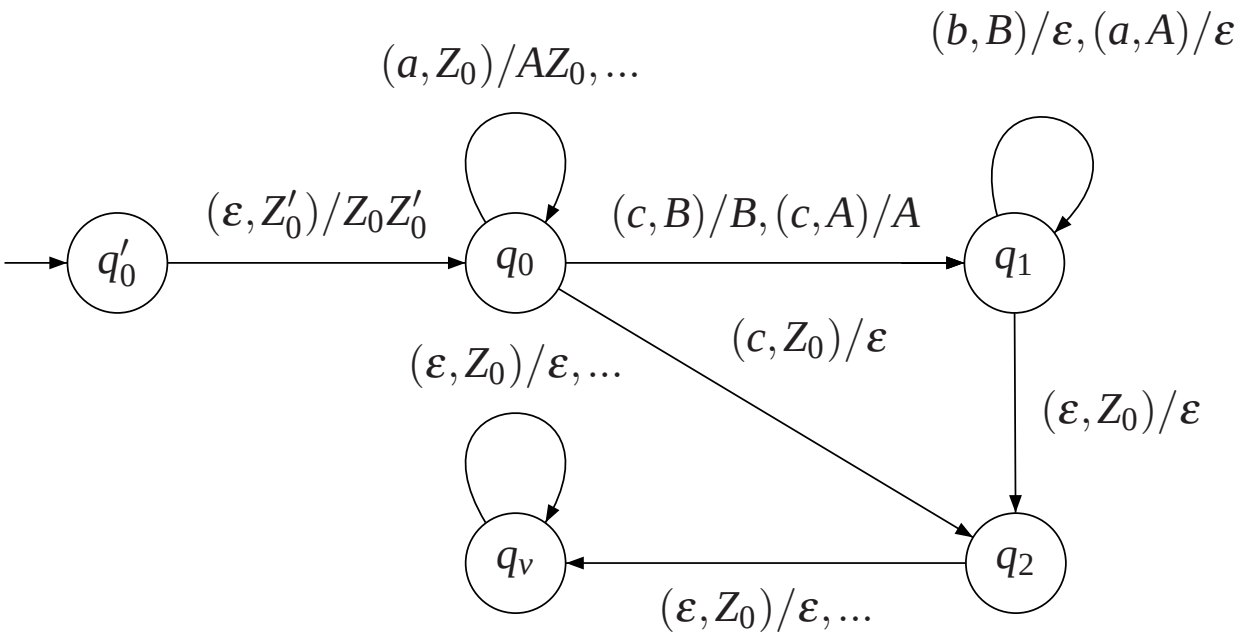

Figura 13: Autômato M' equivalente com critério de aceitação de pilha vazia

Pilha vazia $\Rightarrow$ estado final

Teorema 8.2 "Seja M um autômato de pilha não-determinístico com critério de aceitação baseado em pilha vazia. Então, é possível definir um autômato de pilha não-determinístico M' com critério de aceitação baseado em estado final, de modo que $V(M) = L(M')$."

O Algorimo 8.2 apresenta um método para se efetuar tal conversão.

Pilha vazia $\Rightarrow$ estado final

Algoritmo 8.2 *"Obtenção de um autômato de pilha baseado em critério de aceitação de estado final a partir de outro baseado em critério de pilha vazia."*

- *Entrada: um autômato de pilha não-determinístico* $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$*, com critério de aceitação baseado em pilha vazia.*
- *Saída: um autômato de pilha não-determinístico* $M' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F)$*, com critério de aceitação baseado em estado final.*

Pilha vazia $\Rightarrow$ estado final

Método:

1. $Q' \leftarrow Q \cup \{q_f, q_0'\}$ *(supondo-se, sem perda de generalidade, que* $Q \cap \{q_f, q_0'\} = \emptyset$*);*
2. $\Gamma' \leftarrow \Gamma \cup \{Z_0'\}$ *(supondo-se, sem perda de generalidade, que* $\Gamma \cap \{Z_0'\} = \emptyset$*);*
3. $F = \{q_f\}$*;*
4. *Função de transição* δ'*:*
   1. $\delta' \leftarrow \emptyset$*;*
   2. $\delta'(q_0', \varepsilon, Z_0') \leftarrow \{(q_0, Z_0 Z_0')\}$*;*
   3. $\delta'(q, \sigma, \phi) \leftarrow \delta(q, \sigma, \phi), \quad \forall q \in Q, \phi \in \Gamma, \sigma \in (\Sigma \cup \{\varepsilon\})$*;*
   4. $\delta'(q, \varepsilon, Z_0') \leftarrow \{(q_f, \varepsilon)\}, \quad \forall q \in Q.$

Exemplo

Exemplo 8.2
Considere o autômato de pilha da Figura 14, cujo critério de aceitação é baseado em pilha vazia.

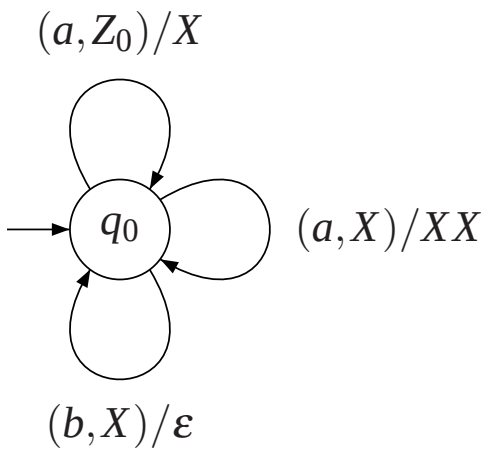

Figura 14: Autômato M com critério de aceitação de pilha vazia

Exemplo

A linguagem aceita por este autômato é constituída pelas sentenças sobre $\{a, b\}$ que contêm a mesma quantidade de símbolos *a* e *b* e, além disso, cuja quantidade acumulada de símbolos *a* seja sempre maior que a quantidade de símbolos *b*, contando-se os símbolos que formam a sentença, da esquerda para a direita. Exemplos de cadeias aceitas são *aaababbb*, *ab* e *abaababb*. Exemplos de cadeias não aceitas são *a*, *ba*, *aaabb* e *abbbbaaa*.

Aplicando-se o Algoritmo 8.2 de conversão do critério de aceitação, obtém-se um novo autômato que reconhece a mesma linguagem e cujo critério é baseado em estado final (Figura 15).

Exemplo

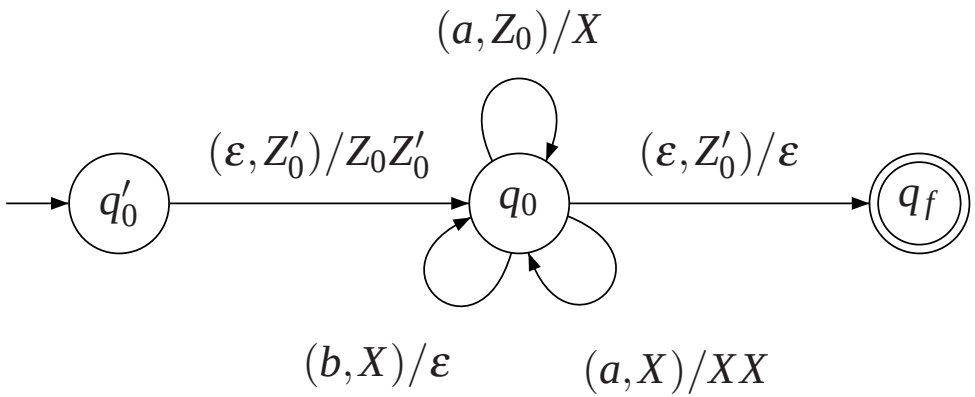

Figura 15: Autômato M' equivalente com critério de aceitação de estado final

Equivalência entre GLCs e APs

Inicialmente, mostra-se que para qualquer gramática livre de contexto é possível definir um autômato de pilha não-determinístico que reconhece exatamente a mesma linguagem gerada pela gramática. A seguir, é apresentado o resultado inverso, ou seja, de que toda e qualquer linguagem aceita por um autômato de pilha não-determinístico pode ser gerada por uma gramática livre de contexto.

GLC $\Rightarrow$ AP

Teorema 9.1 "Seja G uma gramática livre de contexto. Então é possível definir um autômato de pilha não-determinístico M, com critério de aceitação baseado em pilha vazia, de modo que $V(M) = L(G)$."

Considere-se $G = (V, \Sigma, P, S)$. O Algoritmo 9.1 mostra como obter o autômato M a partir de G.

GLC $\Rightarrow$ AP

Algoritmo 9.1 *"Obtenção de um autômato de pilha não-determinístico a partir de uma gramática livre de contexto qualquer."*

- *Entrada: uma gramática livre de contexto* $G = (V, \Sigma, P, S)$*;*
- *Saída: um autômato de pilha não-determinístico* $M = (\{q\}, \Sigma, V, \delta, q, S, \emptyset)$ *com critério de aceitação de pilha vazia, tal que* $V(M) = L(G)$*;*
- *Método:*
  1. *Função de transição:*
     1. $\delta \leftarrow \emptyset$*;*
     2. $\delta(q, \varepsilon, A) = \{(q, \gamma) \mid A \rightarrow \gamma \in P\}, \forall A \in N, \gamma \in V^*$*;*
     3. $\delta(q, \sigma, \sigma) = \{(q, \varepsilon)\}, \forall \sigma \in \Sigma$*.*

GLC $\Rightarrow$ AP

- Note que $Q = \{q\}$, $\Gamma = V$, $Z_0 = S$ e $F = \emptyset$;
- Autômatos construídos segundo este critério operam através da repetida substituição dos símbolos não-terminais no topo da pilha, sem consumo de símbolos da fita de entrada, até que surja um símbolo terminal do topo da pilha;
- Nesta configuração, a sua remoção é condicionada à presença do mesmo símbolo na posição de leitura correntemente apontada pelo cursor da fita de entrada;
- Autômatos construídos segundo este critério também simulam a seqüência de derivações mais à esquerda que seria feita pela gramática correspondente na geração da mesma sentença.

Exemplo

Exemplo 9.1
Considere-se a gramática das expressões aritméticas:

$$\begin{aligned} \{E &\rightarrow T \mid T + E, \\ T &\rightarrow F \mid F * T, \\ F &\rightarrow (E) \mid a\} \end{aligned}$$

Exemplo

Aplicando-se o Algoritmo 9.1, obtém-se o autômato de pilha não-determinístico cuja função de transição δ é:

$$
\begin{aligned}
\{(q,\varepsilon,E) &\rightarrow \{(q,T),(q,T+E)\},\\
(q,\varepsilon,T) &\rightarrow \{(q,F),(q,F*T)\},\\
(q,\varepsilon,F) &\rightarrow \{(q,(E)),(q,a)\},\\
(q,a,a) &\rightarrow \{(q,\varepsilon)\},\\
(q,(,() &\rightarrow \{(q,\varepsilon)\},\\
(q,),)) &\rightarrow \{(q,\varepsilon)\},\\
(q,+,+) &\rightarrow \{(q,\varepsilon)\},\\
(q,*,*) &\rightarrow \{(q,\varepsilon)\}\}
\end{aligned}
$$

Exemplo

Entre as várias possibilidades de movimentação que esse autômato possui para a cadeia de entrada $a+a*a$, e que simulam derivações mais à esquerda na gramática, a seqüência abaixo conduz o autômato à aceitação da mesma:
$(q,a+a*a,E) \Rightarrow (q,a+a*a,T+E) \Rightarrow (q,a+a*a,F+E) \Rightarrow (q,a+a*a,a+E) \Rightarrow$
$(q,+a*a,+E) \Rightarrow (q,a*a,E) \Rightarrow (q,a*a,T) \Rightarrow (q,a*a,F*T) \Rightarrow (q,a*a,a*T) \Rightarrow$
$(q,*a,*T) \Rightarrow (q,a,T) \Rightarrow (q,a,F) \Rightarrow (q,a,a) \Rightarrow (q,\varepsilon,\varepsilon)$

AP $\Rightarrow$ GLC

Teorema 9.2 "Seja M um autômato de pilha com critério de aceitação de pilha vazia. Então é possível definir uma gramática livre de contexto G, de modo que $L(G) = V(M)$."

Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$. A gramática $G = (V, \Sigma, P, S)$, tal que $L(G) = V(M)$, pode ser obtida pela aplicação do Algoritmo 9.2.

AP $\Rightarrow$ GLC

Algoritmo 9.2 *"Obtenção de uma gramática livre de contexto a partir de um autômato de pilha."*

- *Entrada: um autômato de pilha* $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ *com critério de aceitação de pilha vazia;*
- *Saída: uma gramática livre de contexto* $G = (V, \Sigma, P, S)$*, tal que* $L(G) = V(M)$*;*
- *Método:*
  1. $N \leftarrow \{[q_i Z q_k] \mid q_i, q_k \in Q \text{ e } Z \in \Gamma\} \cup \{S\}$;
  2. $P \leftarrow \emptyset$;
  3. $\forall q, q_1 \in Q, \sigma \in \Sigma \cup \{\varepsilon\}, Z, Z_1, \ldots, Z_k \in \Gamma$, *se* $\{(q_1, Z_1 \ldots Z_k)\} \subseteq \delta(q, \sigma, Z)$, $k \geqslant 0$, *então:*
     $P \leftarrow P \cup \{[q Z q_{k+1}] \rightarrow \sigma [q_1 Z_1 q_2][q_2 Z_2 q_3] \ldots [q_k Z_k q_{k+1}]\}$
     para toda e qualquer seqüência de estados $q_2, q_3, \ldots, q_k, q_{k+1}$ *que possa ser obtida a partir de Q (repetições são permitidas);*
  4. $P \leftarrow P \cup \{S \rightarrow [q_0 Z_0 q]\}, \forall q \in Q$.

AP $\Rightarrow$ GLC

É possível demonstrar formalmente (ver Hopcroft79), por indução sobre a quantidade de derivações efetuadas através de G, que a gramática assim definida gera exatamente a mesma linguagem aceita pelo autômato, ou seja:

$$(q_0, x, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \quad \text{se e somente se} \quad S \Rightarrow^* x$$

Exemplo

Exemplo 9.2

Seja $M = (\{q_0, q_1\}, \{a, b\}, \{X, Z_0\}, \delta, q_0, Z_0, \emptyset)$, representado na Figura 16.

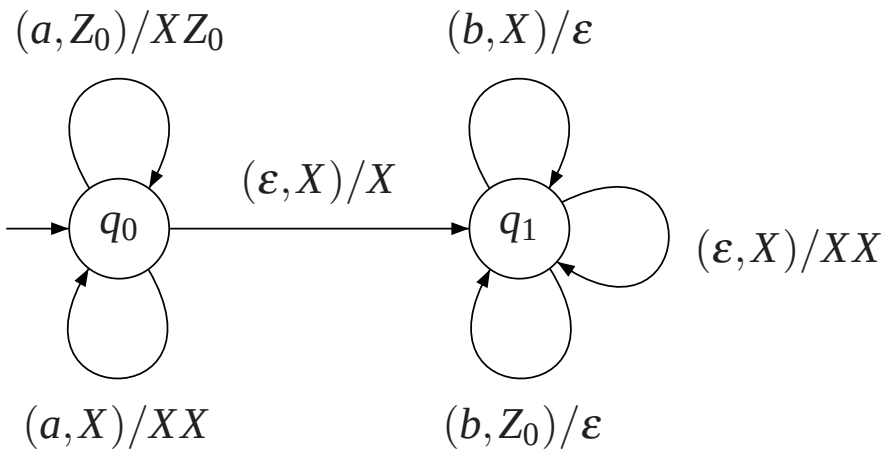

Figura 16: Autômato de pilha M do Exemplo 9.2

Exemplo

A obtenção de G, tal que $L(G) = V(M)$, tem como ponto de partida as regras:

$$\begin{aligned} S &\rightarrow [q_0 Z_0 q_0] \\ S &\rightarrow [q_0 Z_0 q_1] \end{aligned}$$

Exemplo

Para $\delta(q_0, a, Z_0) = (q_0, XZ_0)$, considerar $[q_0 Z_0 \_\_] \rightarrow a[q_0 X \_\_][\_\_ Z_0 \_\_]$ com as listas (q_0, q_0), (q_0, q_1), (q_1, q_0) e (q_1, q_1), gerando:

$$
\begin{aligned}
[q_0 Z_0 q_0] &\rightarrow a[q_0 X q_0][q_0 Z_0 q_0] \\
[q_0 Z_0 q_1] &\rightarrow a[q_0 X q_0][q_0 Z_0 q_1] \\
[q_0 Z_0 q_0] &\rightarrow a[q_0 X q_1][q_1 Z_0 q_0] \\
[q_0 Z_0 q_1] &\rightarrow a[q_0 X q_1][q_1 Z_0 q_1]
\end{aligned}
$$

Exemplo

Para $\delta(q_0, a, X) = (q_0, XX)$, considerar $[q_0 X \_\_] \rightarrow a[q_0 X \_\_][\_\_ X \_\_]$ com as listas (q_0, q_0), (q_0, q_1), (q_1, q_0) e (q_1, q_1), gerando:

$$
\begin{aligned}
[q_0 X q_0] &\rightarrow a[q_0 X q_0][q_0 X q_0] \\
[q_0 X q_1] &\rightarrow a[q_0 X q_0][q_0 X q_1] \\
[q_0 X q_0] &\rightarrow a[q_0 X q_1][q_1 X q_0] \\
[q_0 X q_1] &\rightarrow a[q_0 X q_1][q_1 X q_1]
\end{aligned}
$$

Exemplo

Para $\delta(q_0, \varepsilon, X) = (q_1, X)$, considerar $[q_0 X \_\_] \rightarrow [q_1 X \_\_]$ com as listas (q_0) e (q_1), gerando:

$$\begin{aligned} [q_0 X q_0] &\rightarrow [q_1 X q_0] \\ [q_0 X q_1] &\rightarrow [q_1 X q_1] \end{aligned}$$

Exemplo

Para $\delta(q_1, b, X) = (q_1, \varepsilon)$:

$$[q_1 X q_1] \quad \rightarrow \quad b$$

Exemplo

Para $\delta(q_1, \varepsilon, X) = (q_1, XX)$, considerar $[q_1 X \_\_] \rightarrow [q_1 X \_\_][\_\_ X \_\_]$ com as listas (q_0, q_0), (q_0, q_1), (q_1, q_0) e (q_1, q_1), gerando:

$$
\begin{array}{lcl}
[q_1 X q_0] & \rightarrow & [q_1 X q_0][q_0 X q_0] \\
[q_1 X q_1] & \rightarrow & [q_1 X q_0][q_0 X q_1] \\
[q_1 X q_0] & \rightarrow & [q_1 X q_1][q_1 X q_0] \\
[q_1 X q_1] & \rightarrow & [q_1 X q_1][q_1 X q_1]
\end{array}
$$

Exemplo

Para $\delta(q_1, b, Z_0) = (q_1, \varepsilon)$:

$$[q_1 Z_0 q_1] \quad \rightarrow \quad b$$

Exemplo

A renomeação dos símbolos não-terminais e o agrupamento das regras produz como resultado o conjunto:

$$
\begin{aligned}
S &\rightarrow A \mid B \\
A &\rightarrow aCA \mid aDE \\
B &\rightarrow aCB \mid aDF \\
C &\rightarrow aCC \mid aDG \mid G \\
D &\rightarrow aCD \mid aDH \mid H \\
F &\rightarrow b \\
G &\rightarrow GC \mid HG \\
H &\rightarrow GD \mid b \mid HH
\end{aligned}
$$

Exemplo

Finalmente, a eliminação de símbolos inacessíveis e inúteis resulta em:

$$
\begin{aligned}
S &\rightarrow B \\
B &\rightarrow aDF \\
D &\rightarrow aDH \mid H \\
F &\rightarrow b \\
H &\rightarrow b \mid HH
\end{aligned}
$$

ou seja, $L(G) = V(M) = \{a^i b^j \mid i \geqslant 1 \text{ e } j > i\}$.

Exemplo

Apresenta-se agora uma particular seqüência de movimentos efetuada por M durante o reconhecimento da sentença *aabbbb*:

$$(q_0, aabbbb, Z_0) \vdash (q_0, abbbb, XZ_0) \vdash (q_0, bbbb, XXZ_0) \vdash (q_1, bbbb, XXZ_0) \vdash$$
$$(q_1, bbb, XZ_0) \vdash (q_1, bbb, XXZ_0) \vdash (q_1, bb, XZ_0) \vdash (q_1, b, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$$

A correspondência entre G e M pode ser ilustrada através da análise da seqüência de derivações mais à esquerda obtida para esta mesma cadeia *aabbbb*, e de sua comparação com a seqüência de movimentos efetuados pelo autômato conforme apresentado acima:

$$\begin{aligned} S \quad \Rightarrow \quad & [q_0 Z_0 q_1] \Rightarrow a[q_0 X q_1][q_1 Z_0 q_1] \Rightarrow aa[q_0 X q_1][q_1 X q_1][q_1 Z_0 q_1] \Rightarrow \\ & aa[q_1 X q_1][q_1 X q_1][q_1 Z_0 q_1] \Rightarrow aab[q_1 X q_1][q_1 Z_0 q_1] \Rightarrow \\ & aab[q_1 X q_1][q_1 X q_1][q_1 Z_0 q_1] \Rightarrow \\ & aabb[q_1 X q_1][q_1 Z_0 q_1] \Rightarrow \\ & aabbb[q_1 Z_0 q_1] \Rightarrow \\ & aabbbb \end{aligned}$$

Exemplo

Observe-se que as formas sentenciais obtidas através de G, neste exemplo, estão diretamente relacionadas às configurações assumidas por G em cada etapa do reconhecimento da cadeia considerada. Note-se, portanto, que G "simula", através de uma seqüência de derivações mais à esquerda, a seqüência de movimentos que conduz M de sua configuração inicial até uma configuração final.

LLCs e LRs

As linguagens regulares constituem um subconjunto próprio das linguagens livres de contexto. De fato, é possível demonstrar que toda linguagem regular é também uma linguagem livre de contexto e, por outro lado, que existem linguagens livres de contexto que não são regulares. Esses resultados serão demonstrados, respectivamente, nos Teoremas 10.1 e 10.2 a seguir.

Regulares $\subseteq$ livres de contexto

Teorema 10.1 "Toda linguagem regular é também uma linguagem livre de contexto."

Prova:
Considere-se L uma linguagem regular qualquer. Então, por definição, existe uma gramática linear à direita G que define L. As regras de G possuem todas, sem exceção, apenas um símbolo não-terminal do lado esquerdo e uma cadeia qualquer pertencente a $(\Sigma \cup \{\varepsilon\})(N \cup \{\varepsilon\})$ do lado direito. Como as gramáticas livres de contexto também exigem um único símbolo não-terminal no lado esquerdo das regras, e, além disso, $(\Sigma \cup \{\varepsilon\})(N \cup \{\varepsilon\}) \subset V^*NV^*$, isso implica que toda gramática linear à direita é também uma gramática livre de contexto. Logo, toda linguagem regular é também uma linguagem livre de contexto.

Regulares $\neq$ livres de contexto

Teorema 10.2 "Existem linguagens livres de contexto que não são regulares."

Prova:
Considere-se, por exemplo, a linguagem $L = \{0^i 1 0^i \mid i \geqslant 1\}$. Conforme demonstrado anteriormente, L não é regular, uma vez que não satisfaz ao "Pumping Lemma" das linguagens regulares. No entanto, a gramática $G = (\{S, 0, 1\}, \{0, 1\}, \{S \to 0S0, S \to 010\}, S)$ gera exatamente L. Logo, L é livre de contexto, porém não é regular. De maneira semelhante, pode-se demonstrar a existência de inúmeras outras linguagens com essa mesma característica.

Livre de contexto não-regular

Conforme definido anteriormente, uma linguagem é dita estritamente livre de contexto se ela for livre de contexto, porém não-regular. A característica desse tipo de linguagens é que elas são geradas apenas por gramáticas que possuam pelo menos um símbolo não-terminal que seja auto-recursivo central e essencial. Assim, é possível gerar, nas formas sentenciais geradas pela gramática, subcadeias da forma:

$$Y \Rightarrow^* \alpha Y \beta, \quad \text{com } \alpha, \beta \in \Sigma^+$$

as quais são responsáveis pelo balanceamento dos termos α e β nas sentenças da linguagem.

Livre de contexto não-regular

De fato, gramáticas lineares à direita não permitem a definição de símbolos não-terminais com essa propriedade. Ao contrário, gramáticas lineares à direita geram apenas formas sentenciais da seguinte forma:

$$Y \Rightarrow^* \alpha Y, \quad \text{com } \alpha \in \Sigma^+$$

em que, obviamente, não há balanceamento de termos nem, portanto, aninhamentos sintáticos. A existência de termos balanceados (ou aninhados) é, por isso, o fator que diferencia uma linguagem estritamente livre de contexto de uma linguagem regular.

Conceitos

- Linguagens regulares podem ser definidas através de expressões regulares, gramáticas regulares ou autômatos finitos;
- Por outro lado, a existência de linguagens não-regulares foi provada através do uso do “Pumping Lemma” das linguagens regulares;
- De maneira análoga, para provar que uma linguagem é livre de contexto, é suficiente apresentar uma gramática livre de contexto que gere esta linguagem, ou ainda um autômato de pilha que a reconheça;
- A existência de linguagens que não são livres de contexto pode ser demonstrada com o auxílio do “Pumping Lemma” das linguagens livres de contexto, apresentado a seguir.

“Pumping Lemma” para linguagens livres de contexto

Teorema 11.1 “Seja L uma linguagem livre de contexto, com $\varepsilon \notin L$. Então, existe uma constante inteira n, dependente apenas de L, que satisfaz às seguintes condições: (i) $\forall \gamma \in L, |\gamma| \geqslant n, \gamma = uvwxy$; (ii) $|vwx| \leqslant (n-1) * 2$; (iii) $|vx| \geqslant 1$; (iv) $\forall i \geqslant 0, uv^i wx^i y \in L$.”

“Pumping Lemma” para linguagens livres de contexto

Se L é livre de contexto e não contém a cadeia vazia, então $L = L(G)$, sendo $G = (V, \Sigma, P, S)$, G uma gramática livre de contexto na Forma Normal de Chomsky. Portanto, qualquer árvore de derivação que represente uma sentença de L nessa gramática será uma árvore binária.
Por outro lado, árvores binárias de altura i geram sentenças de comprimento máximo 2^{i-1}. A Figura 17 ilustra essa relação para os casos i=1, 2 e 3, e facilita a generalização da mesma.

“Pumping Lemma” para linguagens livres de contexto

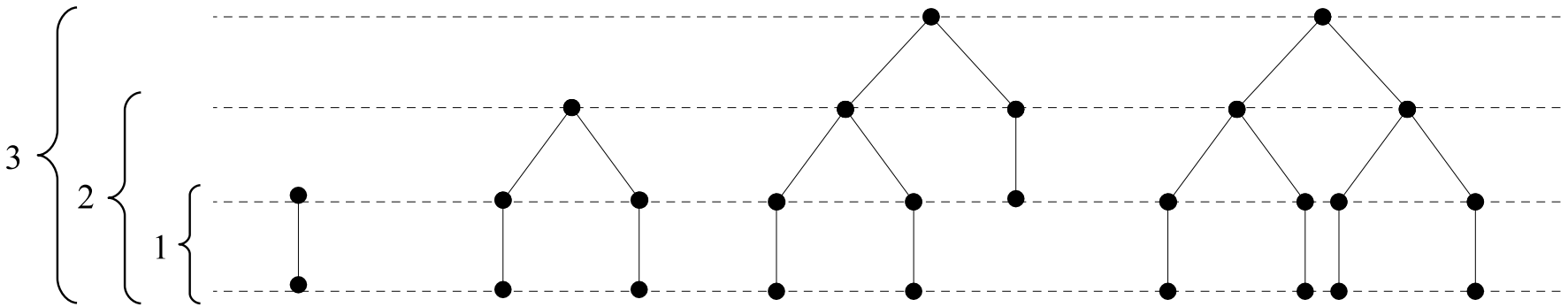

Figura 17: Altura e comprimento de sentenças em árvores binárias geradas por gramáticas na Forma Normal de Chomsky

“Pumping Lemma” para linguagens livres de contexto

- Através desta figura, é fácil perceber que árvores de altura 1 geram sentenças de comprimento 1 (portanto $\leqslant 2^0$), árvores de altura 2 geram sentenças de comprimento 2 ($\leqslant 2^1$) e árvores de altura 3 geram sentenças de comprimento 3 ou 4 ($\leqslant 2^2$);
- No caso geral, árvores com altura i geram sentenças de comprimento menor ou igual a 2^{i-1};
- Em outras palavras, árvores com altura $i+1$ geram sentenças de comprimento máximo 2^i;
- Logo, se uma sentença tem comprimento mínimo (maior ou igual a) 2^i, então a árvore de derivação correspondente possuirá altura mínima (maior ou igual a) $i+1$;
- De fato, temos que (i) é necessária uma árvore com altura pelo menos $i+1$ para gerar uma sentença de comprimento 2^i e (ii) são necessárias árvores com alturas maiores para sentenças ainda mais longas.

“Pumping Lemma” para linguagens livres de contexto

Considere-se $k = |N|$, onde N é o conjunto dos símbolos não-terminais de G, estando esta expressa na Forma Normal de Chomsky, e faça-se $n = 2^{k-1} + 1$.

“Pumping Lemma” para linguagens livres de contexto

- Alguns autores adotam $n = 2^k$;
- Nesse caso, as condições do enunciado tornam-se: (i) $\forall \gamma \in L, |\gamma| \geqslant n, \gamma = uvwxy$; (ii) $|vwx| \leqslant n$; (iii) $|vx| \geqslant 1$; (iv) $\forall i \geqslant 0, uv^i wx^i y \in L$, o que pode ser provado de maneira similar à presente demonstração (basta notar que se $|\gamma| \geqslant 2^k$, então da mesma forma a árvore de derivação terá altura mínima $k+1$;
- Essa versão corresponde ao enunciado mais comum do presente teorema, sendo usada na maioria das aplicações, como é o caso dos exemplos seguintes, por causa da sua praticidade ao igualar o comprimento mínimo da sentença γ com o comprimento máximo da cadeia $|vwx|$.

“Pumping Lemma” para linguagens livres de contexto

Se $\gamma \in L$ e $|\gamma| \geqslant n$, isto é, se $|\gamma| \geqslant 2^{k-1}+1$, então, face ao resultado anterior, é certo que a altura da árvore de derivação correspondente à sentença γ será maior ou igual a $k+1$ (pois com árvores de altura k é possível apenas gerar cadeias de comprimento máximo 2^{k-1}). Suponha-se que o valor desta altura seja p (portanto, $p \geqslant k+1$) e considere-se um caminho z qualquer na árvore que possua comprimento p (haverá pelo menos um caminho que satisfaça a essa condição).

“Pumping Lemma” para linguagens livres de contexto

Se o caminho selecionado possui comprimento p, então este caminho é formado de q símbolos, $q \geqslant k+2$. Desses q símbolos, apenas um será terminal (o último símbolo do caminho, aquele que é folha da árvore) e os demais serão necessariamente não-terminais.

“Pumping Lemma” para linguagens livres de contexto

Ignorando o símbolo terminal e concentrando a atenção nos $q-1$ não-terminais, seus antecessores, se $q \geqslant k+2$, então existem r símbolos não-terminais neste caminho, e $r = q-1$, ou seja, $r \geqslant k+1$.

“Pumping Lemma” para linguagens livres de contexto

Considerem-se agora apenas os primeiros $k+1$ símbolos não-terminais consecutivos que antecedem imediatamente a folha da árvore, ignorando os $r-(k+1)$ símbolos não-terminais situados no início do caminho selecionado. O caminho escolhido z pode, portanto, ser considerado como:

$$z = \mu\rho\sigma, \text{ com } \mu \in N^*, |\mu| = r-(k+1), \rho \in N^+, |\rho| = k+1 \text{ e } \sigma \in \Sigma$$

“Pumping Lemma” para linguagens livres de contexto

A Figura 18 ilustra esta interpretação do caminho z.

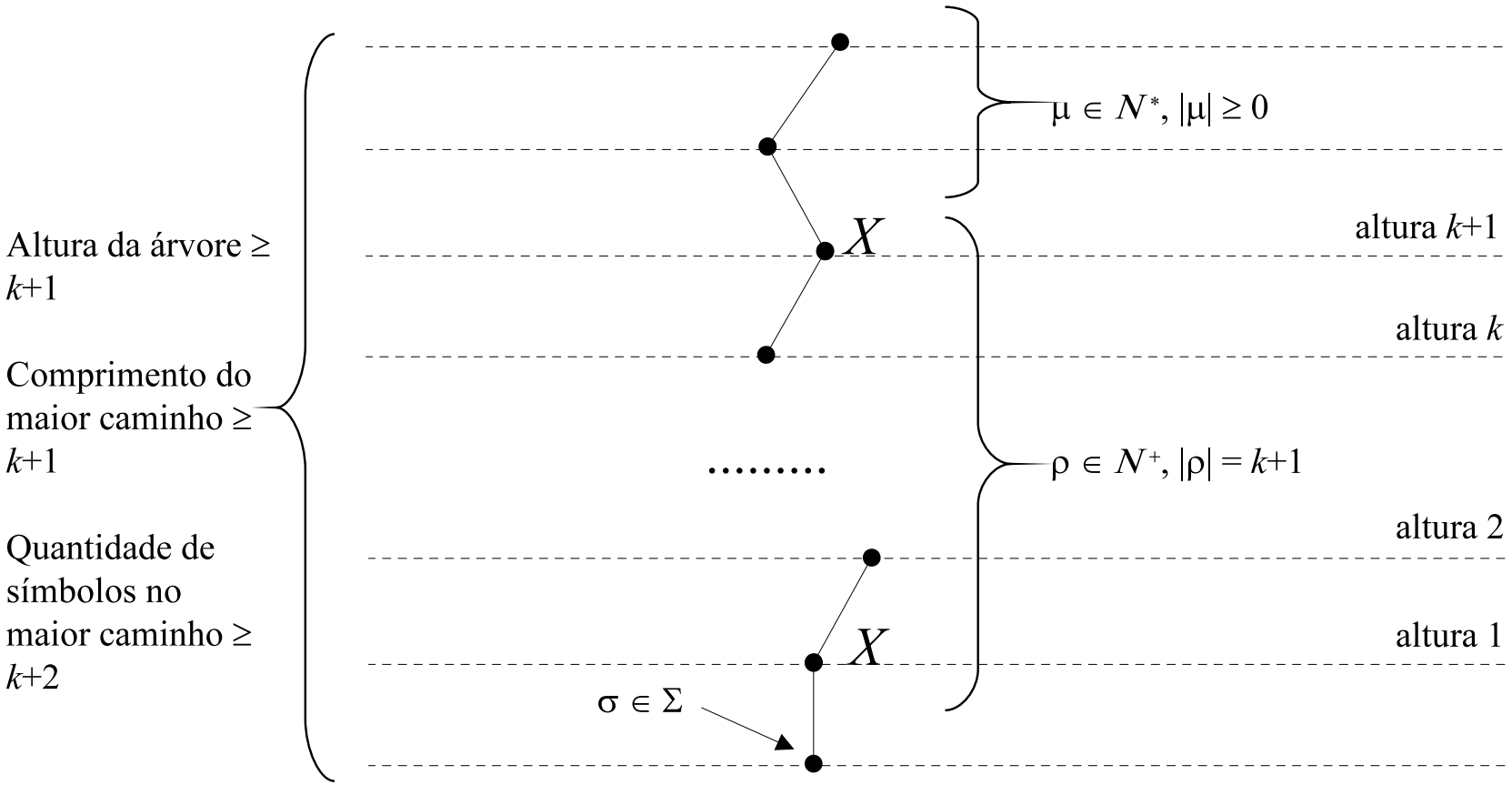

Figura 18: Múltiplas ocorrências do símbolo não-terminal X no caminho z

“Pumping Lemma” para linguagens livres de contexto

Se ρ contém exatamente $k+1$ símbolos não-terminais e a gramática contém apenas k símbolos não-terminais distintos, é certo que pelo menos um símbolo não-terminal de G aparece mais de uma vez em ρ. Se chamarmos a este símbolo X, as seguintes considerações poderão ser feitas acerca das posições em que os símbolos X podem aparecer na árvore de derivação:

“Pumping Lemma” para linguagens livres de contexto

1. Para o *primeiro* X (aquele que está na posição mais alta da árvore, próxima da raiz):
   - Ele pode ser o primeiro símbolo da cadeia ρ (portanto, estar na altura $k+1$);
   - Ele pode ser o penúltimo símbolo da cadeia ρ (portanto, estar na altura 2), uma vez que o segundo X comparece por último nesta mesma cadeia;
   - Ele pode assumir qualquer posição entre essas duas.
2. Para o *segundo* X (aquele que está na posição mais baixa da árvore, próxima da folha):
   - Ele pode ser o segundo símbolo da cadeia ρ (portanto, estar na altura k), uma vez que o primeiro X comparece antes nesta mesma cadeia;
   - Ele pode ser o último símbolo da cadeia ρ (portanto, na altura 1);
   - Ele pode assumir qualquer posição entre essas duas.

“Pumping Lemma” para linguagens livres de contexto

Assim, pode-se concluir:

1. Para o *primeiro* X: se $X \Rightarrow^* \alpha_1, \alpha_1 \in \Sigma^*$, então $2 \leqslant |\alpha_1| \leqslant 2^k$;
2. Para o *segundo* X: se $X \Rightarrow^* \alpha_2, \alpha_2 \in \Sigma^*$, então $1 \leqslant |\alpha_2| \leqslant 2^{k-1}$.

A situação da árvore de derivação da sentença γ pode ser representada como mostra a Figura 19 (S representa a raiz de G, e pode, eventualmente, coincidir com o primeiro símbolo da cadeia ρ).

“Pumping Lemma” para linguagens livres de contexto

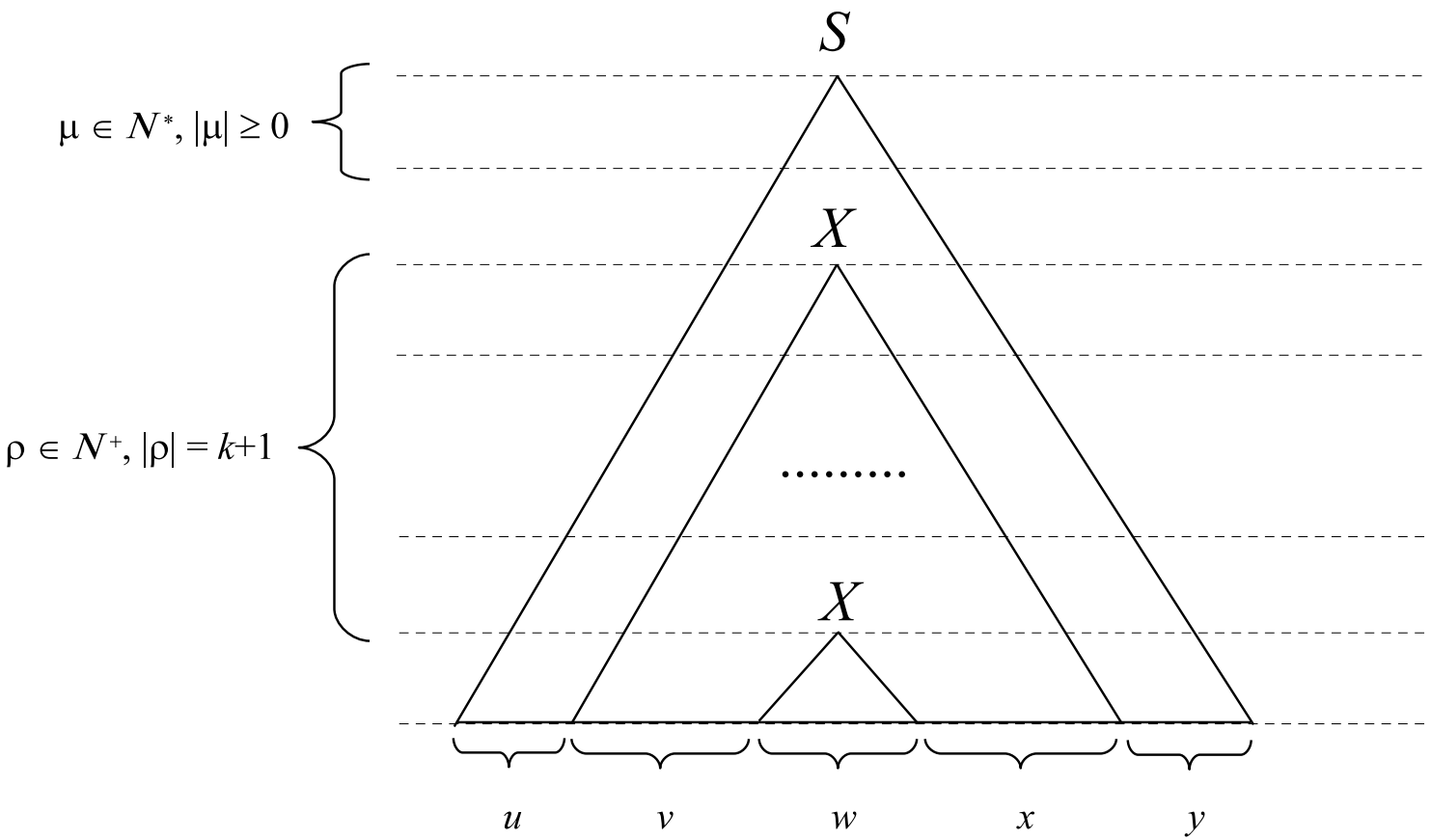

Figura 19: Árvore de derivação da cadeia $\gamma = uvwxy$

“Pumping Lemma” para linguagens livres de contexto

A inspeção desta árvore nos permite chegar às seguintes conclusões:

- A sentença γ pode ser considerada como sendo composta por cinco partes, $\gamma = uvwxy$;
- Como $w = \alpha_2$, então $1 \leqslant |w| \leqslant 2^{k-1}$;
- Como $vwx = \alpha_1$, então $2 \leqslant |vwx| \leqslant 2^k$ e $2 \leqslant |vwx| \leqslant (n-1) * 2$, pois $(n-1) * 2 = ((2^{k-1}+1)-1) * 2 = 2^{k-1} * 2 = 2^k$. Observar, nesse caso, que se $n = 2^k$, então $|vwx| \leqslant n$;
- Como $1 \leqslant |w|$ e $2 \leqslant |vwx|$, então $|vx| \geqslant 1$, ou seja, v e x não podem ser ambas vazias.

“Pumping Lemma” para linguagens livres de contexto

Além disso, as regras de G permitem a derivação das seguintes formas sentenciais:

- $S \Rightarrow^* uXy$
- $X \Rightarrow^* vXx$
- $X \Rightarrow^* w$

“Pumping Lemma” para linguagens livres de contexto

A inspeção dessas formas sentenciais, assim como a análise da árvore mostrada na Figura 19, permite concluir que existem outras possibilidades de derivações (ou seja, de construção da árvore) de sentenças em G, as quais produzem sentenças diversas da original γ, e que, por construção, devem necessariamente pertencer à linguagem L.

“Pumping Lemma” para linguagens livres de contexto

Por exemplo, é possível imaginar que, em vez de aplicar regras que fazem o primeiro X derivar vXx, é possível aplicar, a esta ocorrência de X, as regras aplicadas ao segundo X, e dessa maneira gerar a sentença uwy no lugar da sentença $uvwxy$. Da mesma forma, seria possível repetir a derivação aplicada ao primeiro X no lugar da derivação feita para o segundo X, e com isso gerar a sentença $uvvwxxy$. Generalizando, todas as sentenças uv^iwx^iy, com $i \geqslant 0$, podem ser derivadas em G, e, portanto, pertencem necessariamente a $L(G)$. A Figura 20 mostra, graficamente, a aplicação desta idéia.

“Pumping Lemma” para linguagens livres de contexto

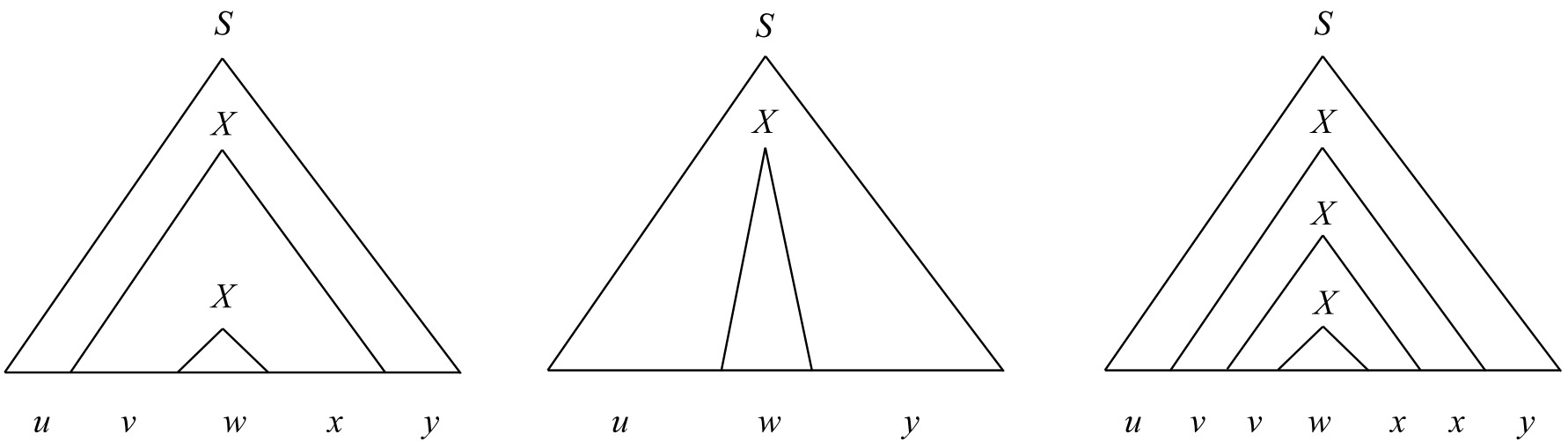

Figura 20: Árvores de derivação para as cadeias $uvwxy$, uwy e $uvvwxxy$

“Pumping Lemma” para linguagens livres de contexto

Em termos gramaticais, como $S \Rightarrow^* uXy$, $X \Rightarrow^* vXx$ e $X \Rightarrow^* w$, ou seja, $S \Rightarrow^* uvwxy$, então, necessariamente:

- $S \Rightarrow^* uXy \Rightarrow^* uwy$
- $S \Rightarrow^* uXy \Rightarrow^* uvvwxxy$
- $S \Rightarrow^* uXy \Rightarrow^* uvvvwxxxy$
- $\ldots$
- $S \Rightarrow^* uXy \Rightarrow^* uv^i wx^i y, \forall i \geqslant 0$

“Pumping Lemma” para linguagens livres de contexto

Considerado de maneira informal, este “Pumping Lemma” para linguagens livres de contexto pode ser interpretado da seguinte maneira:

- Sentenças que apresentem um certo comprimento mínimo geram àrvores de derivação com uma certa altura mínima;
- Como a quantidade de símbolos não-terminais da gramática é limitada, necessariamente deverá haver repetição de símbolos em algum caminho da árvore de derivação correspondente;
- Tal repetição permite a geração de uma quantidade infinita de outras sentenças estruturalmente similares, que devem necessariamente pertencer à linguagem.

“Pumping Lemma” para linguagens livres de contexto

O símbolo não-terminal X, na demonstração acima, possibilita derivações do tipo $X \Rightarrow^* vXx$ e corresponde ao símbolo que denominamos previamente de auto-recursivo central essencial. A existência de pelo menos um símbolo desse tipo na gramática garante a possibilidade de se gerar uma quantidade infinita de outras sentenças que também pertençam à mesma linguagem.

“Pumping Lemma” para linguagens livres de contexto

Se v ou x forem vazios, a linguagem será regular (note-se que o “Pumping Lemma” para linguagens regulares é um caso particular do “Pumping Lemma” para linguagens livres de contexto). Se v e x forem simultaneamente diferentes da cadeia vazia, então a linguagem é livre de contexto e não-regular, e os termos v e x ocorrerão sempre de forma balanceada nas sentenças da linguagem.

Exemplo

Exemplo 11.1
A linguagem $\{a^i b a^i \mid i \geqslant 0\}$ é gerada pela gramática livre de contexto $G = (\{S,X,Y,Z,a,b\},\{a,b\},\{S \to XY \mid b, X \to ZS, Z \to a, Y \to a\},S)$, que está na Forma Normal de Chomsky e possui $N = \{S,X,Y,Z\}$, $|N| = 4$. De acordo com a demonstração do "Pumping Lemma", a constante n para G é $2^{k-1}+1 = 2^{|N|-1}+1 = 2^{4-1}+1 = 2^3+1 = 8+1 = 9$.
A cadeia $z = a^4 b a^4 \in L(G)$ é tal que $|z| \geqslant 9$. Portanto, $z = uvwxy$, com $|vwx| \leqslant n$ e $|vx| \geqslant 1$. A Figura 21 apresenta a árvore de derivação correspondente à cadeia z.
A árvore da Figura 21 possui altura 9 (portanto maior ou igual a $|N|+1 = 5$) e contém diversos caminhos formados por $|N|+2 = 6$ ou mais símbolos, nos quais é possível isolar os $|N|+1$ últimos símbolos não-terminais.

Exemplo

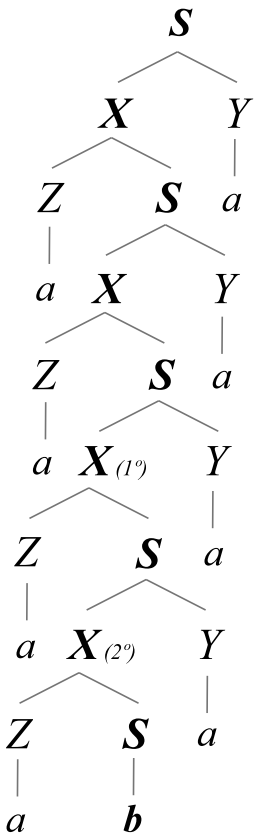

Figura 21: Árvore de derivação para a sentença a^4ba^4

Exemplo

Considere-se o caminho $SXSXSXSXSb$, cujo comprimento é 10, e os 5 últimos símbolos não-terminais que o compõe, $SXSXS$. De acordo com o "Pumping Lemma", esse caminho deve conter (e contém) pelo menos um símbolo não-terminal repetido: o símbolo S ocorre três vezes e o símbolo X ocorre duas vezes.
Considere-se o símbolo X. A primeira ocorrência do mesmo gera a cadeia $aaba$, ao passo que a segunda ocorrência gera a cadeia ab. Logo:

$$z = \underbrace{aa}_{u}\underbrace{a}_{v}\underbrace{ab}_{w}\underbrace{a}_{x}\underbrace{aaa}_{y}$$

Exemplo

Conseqüentemente, as cadeias uv^iwx^iy, com $i \geqslant 0$, devem também pertencer à linguagem. De fato, as cadeias uwy e uv^2wx^2y pertencem a $L(G)$:

$$\underbrace{aa}_{u}\underbrace{ab}_{w}\underbrace{aaa}_{y} = a^3ba^3$$

$$\underbrace{aa}_{u}\underbrace{a}_{v}\underbrace{a}_{v}\underbrace{ab}_{w}\underbrace{a}_{x}\underbrace{a}_{x}\underbrace{aaa}_{y} = a^5ba^5$$

A cadeia uwy pode ser obtida substituindo-se as derivações aplicadas ao primeiro símbolo X pelas aplicadas ao segundo símbolo X. A nova seqüência de derivações torna-se (ver Figura 22):

$$S \Rightarrow^* aa\underbrace{X}_{1^o}aaa \Rightarrow aa\underbrace{ab}_{w}aaa = a^3ba^3$$

Exemplo

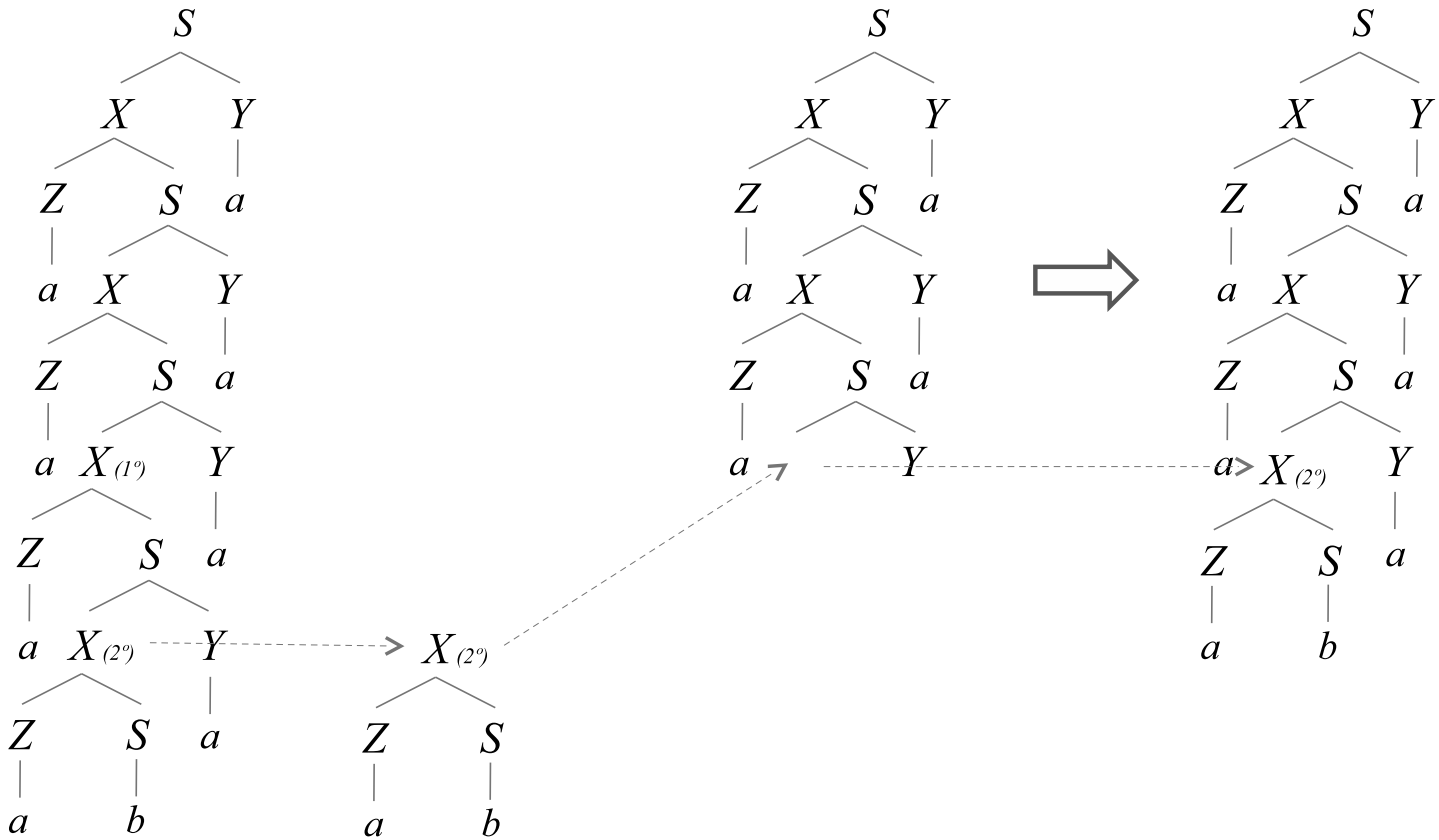

Figura 22: Árvore de derivação para a sentença a^3ba^3

Exemplo

A cadeia uv^2wx^2y pode ser obtida substituindo-se as derivações aplicadas ao segundo símbolo X pelas aplicadas ao primeiro símbolo X. A nova seqüência de derivações torna-se (ver Figura 23):

$$S \Rightarrow^* aaa\underbrace{X}_{2^o}aaaa \Rightarrow aaa\underbrace{aaba}_{vwx}aaaa = a^5ba^5$$

Exemplo

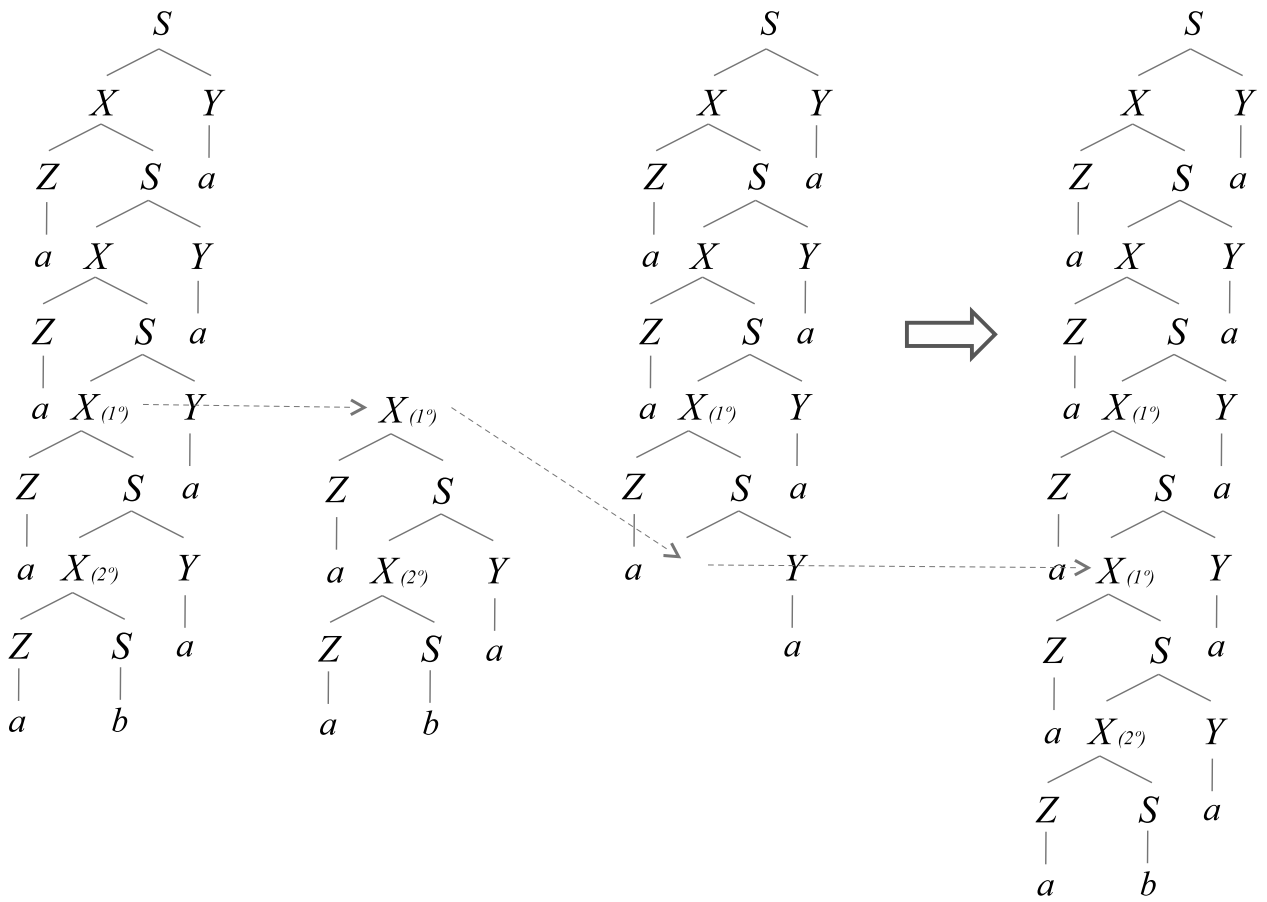

Figura 23: Árvore de derivação para a sentença a^5ba^5

Exemplo

A iteração do último passo (Figura 23) possibilita a geração das sentenças uv^iwx^iy, com $i \geqslant 3$.

Exemplo

Como no caso do “Pumping Lemma” para linguagens regulares, que estabelece uma propriedade inerente a toda e qualquer linguagem regular, o “Pumping Lemma” para linguagens livres de contexto estabelece uma propriedade inerente a toda e qualquer linguagem livre de contexto. De maneira análoga àquele caso, uma de suas principais aplicações é na demonstração da existência de linguagens que não são livres de contexto, conforme mostrarão os exemplos a seguir.

Observe-se que o teorema garante a existência da constante n, mas a aplicação do mesmo não exige que seja determinado o seu valor, como mostram os exemplos seguintes.

Exemplo

Exemplo 11.2
A linguagem $L_1 = \{a^k b^k c^k \mid k \geqslant 1\}$ não é livre de contexto. Suponha-se, por hipótese, que L_1 seja livre de contexto. De acordo com o Teorema 11.1 ("Pumping Lemma" para linguagens livres de contexto), existe uma constante inteira n tal que, qualquer que seja a sentença $\gamma \in L_1$, $|\gamma| \geqslant n$, então $\gamma = uvwxy$, $|vwx| \leqslant n$, $|vx| \geqslant 1$ e $uv^i wx^i y \in L_1$.

Exemplo

Seja $\gamma = a^n b^n c^n$. Como $|a^n b^n c^n| = 3 * n$, então está satisfeita a condição $|\gamma| \geqslant n$. Portanto, $\gamma = uvwxy$. Nesta situação, a subcadeia vwx pode assumir um dos seguintes formatos (lembrar que $|vwx| \leqslant n$):

1. vwx contém apenas símbolos "a" (pelo menos um); portanto, vx também contém apenas símbolos a (pelo menos um);
2. vwx contém apenas símbolos "b" (pelo menos um); portanto, vx também contém apenas símbolos b (pelo menos um);
3. vwx contém apenas símbolos "c" (pelo menos um); portanto, vx também contém apenas símbolos c (pelo menos um);
4. vwx contém símbolos "a" (pelo menos um) seguidos de símbolos "b" (pelo menos um); portanto, vx contém pelo menos um símbolo a ou b, porém nenhum símbolo c;
5. vwx contém símbolos "b" (pelo menos um) seguidos de símbolos "c" (pelo menos um); portanto, vx contém pelo menos um símbolo b ou c, porém nenhum símbolo a.

Exemplo

É impossível que a subcadeia vwx contenha simultaneamente símbolos "a", "b" e "c", uma vez que, para isso acontecer, seria necessário que o comprimento de vwx fosse no mínimo $n+2$, o que contraria o "Pumping Lemma".

Exemplo

Passando-se à análise de cada uma dessas possibilidades, considerando o formato assumido pela sentença *uwy*, isto é, pela sentença *uvwxy* após a remoção das subcadeias *v* e *x*, o que é previsto pelo "Lemma" quando estabelece que todas as sentenças uv^iwx^iy devem pertencer a L_1 (neste caso, faz-se $i = 0$). Lembrar que $|vx| \geqslant 1$:

1. *uwy* conterá *n* símbolos "*b*", *n* símbolos "*c*" e uma quantidade de símbolos "*a*" menor que *n*; logo, *uwy* não pertence a L_1;
2. *uwy* conterá *n* símbolos "*a*", *n* símbolos "*c*" e uma quantidade de símbolos "*b*" menor que *n*; logo, *uwy* não pertence a L_1;
3. *uwy* conterá *n* símbolos "*a*", *n* símbolos "*b*" e uma quantidade de símbolos "*c*" menor que *n*; logo, *uwy* não pertence a L_1;

Exemplo

4. *uwy* conterá *n* símbolos "*c*", e quantidades de símbolos "*a*" e de símbolos "*b*" (pelo menos uma delas) respectivamente menores que *n*; logo, *uwy* não pertence a L_1;
5. *uwy* conterá *n* símbolos "*a*", e quantidades de símbolos "*b*" e de símbolos "*c*" (pelo menos uma delas) respectivamente menores que *n*; logo, *uwy* não pertence a L_1.

Exemplo

Portanto, a sentença $a^n b^n c^n$ contradiz a hipótese inicial, provando que a linguagem L_1 não é livre de contexto. Cumpre observar que, anteriormente ("Pumping Lemma" para as Linguagens Regulares), esta mesma linguagem foi demonstrada como sendo não-regular.

Exemplo

Exemplo 11.3
A linguagem $L_2 = \{wcw \mid w \in \{a,b\}^+\}$ não é livre de contexto. Esta linguagem sintetiza uma característica bastante comum nas linguagens de programação mais usuais: a necessidade de se usarem identificadores idênticos em partes diferentes de um mesmo programa, por exemplo, na declaração de uma variável e na ocasião de sua utilização posterior.
Suponha-se, por hipótese, que L_2 seja livre de contexto. De acordo com o "Pumping Lemma" para linguagens livres de contexto, existe uma constante inteira n tal que, qualquer que seja a sentença $\gamma \in L_2$, $|\gamma| \geqslant n$, então $\gamma = uvwxy$, $|vwx| \leqslant n$, $|vx| \geqslant 1$ e $uv^iwx^iy \in L_2$.

Exemplo

Considere-se, por exemplo, a sentença $\gamma = a^n b^n c a^n b^n$. Como $|\gamma| = 4 * n + 1 \geqslant n$, então está satisfeita a condição do "Pumping Lemma" para a escolha da sentença γ. Portanto, $\gamma = uvwxy$ e a subcadeia vwx pode assumir um dos seguintes formatos:

1. vwx contém apenas símbolos "a" (pelo menos um);
2. vwx contém apenas símbolos "b" (pelo menos um);
3. vwx contém símbolos "a" (pelo menos um) seguidos de símbolos "b" (pelo menos um);
4. vwx se inicia com símbolos "b" (zero ou mais) e termina com o símbolo "c";
5. vwx se inicia com o símbolo "c" e termina com símbolos "a" (zero ou mais);
6. vwx se inicia com símbolos "b" (pelo menos um) , continua com um símbolo "c" e termina com símbolos "a" (pelo menos um).

Exemplo

De maneira análoga ao que foi mostrado no Exemplo 11.2, passa-se a examinar o formato das cadeias uwy que são geradas em cada um desses casos:

1. Ao extrair símbolos "a" da subcadeia w que antecede o símbolo "c", esta resulta diferente da subcadeia w posterior ao mesmo "c"; o mesmo acontece se forem extraídos símbolos "a" da subcadeia posterior ao símbolo "c"; logo, uwy não pertence a L_2;
2. Semelhante ao caso anterior;
3. Modifica, exclusivamente, a subcadeia anterior ao símbolo "c" ou a subcadeia posterior ao símbolo "c"; logo, uwy não pertence a L_2;
4. Como $|vx| \geqslant 1$, vx contém pelo menos um símbolo "b" ou um símbolo "c", que, se removidos da sentença γ, provocam a geração de uma sentença que não pertence a L_2 (seja porque as cadeias w tornam-se diferentes, seja porque o símbolo "c" que as separa desaparece);
5. Semelhante ao caso anterior;
6. Como $|vx| \geqslant 1$, vx contém pelo menos um símbolo "b" (da primeira subcadeia w) ou um símbolo "c" ou, ainda, um símbolo "a" (da segunda subcadeia w); qualquer que seja o caso, a cadeia resultante uwy não pertence a L_2 (seja porque as cadeias w tornam-se diferentes, seja porque o símbolo "c" que as separa desaparece).

Exemplo

Pelo exposto, fica claro que a hipótese inicial é falsa e que L_2 não é uma linguagem livre de contexto.

Exemplo

Exemplo 11.4
A linguagem $L_3 = \{a^k | k \geqslant 1$ é um número primo$\}$ não é livre de contexto.
Suponha-se que L_3 seja livre de contexto e considere-se a sentença $\gamma = a^p$, $p \geqslant n+2$, onde é o valor da constante n definida pelo "Pumping Lemma" para linguagens livres de contexto. Se $\gamma = uvwxy$ pertence a L_3, então, de acordo com o "Lemma", a sentença uwy também deve pertencer. Seja $q = |uwy|$.
O comprimento das sentenças uv^iwx^iy pode ser calculado da seguinte forma: $|uv^iwx^iy| = |uwy| + i * |vx|$. Em particular, o comprimento da sentença $|uv^qwx^qy| = |uwy| + q * |vx| = q + q * |vx| = q * (1 + |vx|)$. Além disso,

- $q = |uwy| = |uvwxy| - |vx|$. Como, pelo "Pumping Lemma", $|vwx| \leqslant n$, segue que $|vx| \leqslant n$. Como $|uvwxy| \geqslant n+2$, segue que $q \geqslant 2$;
- Pelo "Pumping Lemma", $|vx| \geqslant 1$. Logo, $1 + |vx| \geqslant 2$.

Exemplo

Portanto, o comprimento de $|uv^q wx^q y|$ corresponde ao produto de dois números maiores que 1 (ou seja, ele não é primo) e isso prova que L_3 não pode ser livre de contexto.

Vale lembrar que, anteriormente ("Pumping Lemma" para as Linguagens Regulares), esta mesma linguagem foi demonstrada como sendo não-regular.

Conceito

- Diferentemente da classe das linguagens regulares, que podem ser reconhecidas indistintamente por autômatos finitos, determinísticos ou não-determinísticos, somente autômatos de pilha não-determinísticos são capazes de reconhecer as linguagens livres de contexto, no caso geral.
- Esta conclusão, que pode ser provada demonstrando-se a existência de linguagens livres de contexto não-reconhecíveis por autômatos de pilha determinísticos, quaisquer que sejam eles, possui uma importante conseqüência prática, uma vez que os autômatos de pilha não-determinísticos são por natureza pouco eficientes, e por isso constituem modelos de implementação pouco atraentes em diversas situações, entre as quais, na importante área de construção de compiladores.

Exemplo

Exemplo 12.1
A linguagem $L_1 = \{ww^R \mid w \in \{a,b\}^*\}$ é não-determinística, ou seja, é reconhecida apenas por autômatos de pilha não-determinísticos. Um exemplo de tal autômato, com critério de aceitação baseado em pilha vazia, é apresentado na Figura 24.

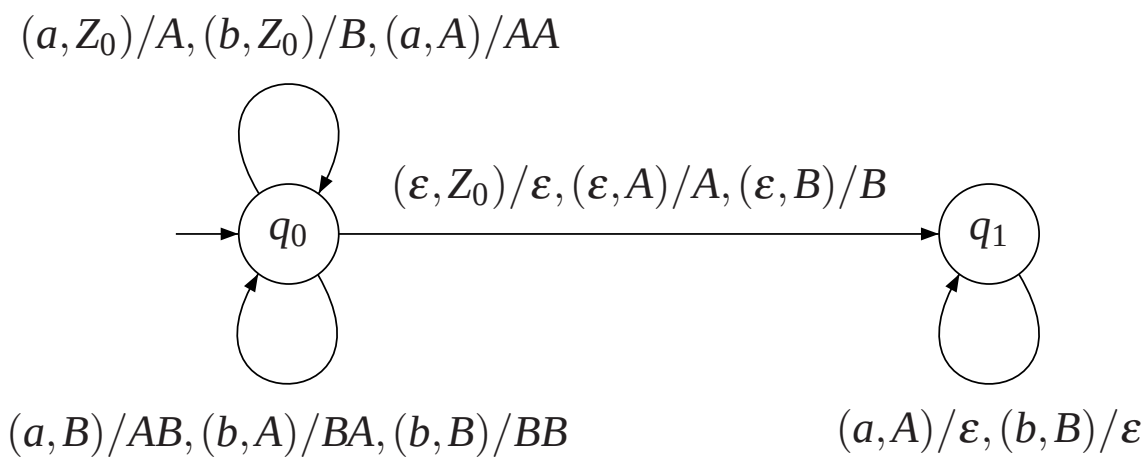

Figura 24: Autômato de pilha para a linguagem $\{ww^R \mid w \in \{a,b\}^*\}$

Exemplo

São exemplos de cadeias aceitas por este autômato *aa* e *abba*, como se pode verificar nos reconhecimentos abaixo:

- $(q_0, aa, Z_0) \vdash (q_0, a, A) \vdash (q_1, a, A) \vdash (q_1, \varepsilon, \varepsilon)$
- $(q_0, abba, Z_0) \vdash (q_0, bba, A) \vdash (q_0, ba, BA) \vdash (q_1, ba, BA) \vdash (q_1, a, A) \vdash (q_1, \varepsilon, \varepsilon)$

Como é fácil observar, a característica não-determinística do autômato é fundamental para que ele possa determinar o ponto exato da cadeia de entrada em que termina a subcadeia w e se inicia a subcadeia w^R: apenas uma movimentação em vazio no ponto central da cadeia será capaz de fazer com que o autômato atinja uma configração final. Tal característica é necessária, uma vez que o autômato, por conta de suas próprias limitações, não tem condições de determinar diretamente o ponto exato em que se inicia a cadeia w^R.

Linguagens livres de contexto e não-determinismos

Assim, apesar do elevado interesse prático exibido pelas linguagens livres de contexto, decorrente da relativa facilidade com que elas podem ser representadas e manipuladas, a dificuldade acima mencionada levou os pesquisadores a um estudo mais aprofundado dessa classe de linguagens, visando fornecer aos projetistas e implementadores de linguagens artificiais subsídios que permitissem o aproveitamento prático das características dessa classe de linguagens, sem no entanto sacrificar o desempenho dos respectivos reconhecedores pela presença de eventuais não-determinismos.

Linguagem livre de contexto determinística

- Como conseqüência, foi identificada e caracterizada a classe das **linguagens livres de contexto determinísticas**, que se provou ser subconjunto próprio das linguagens livres de contexto genéricas;
- Por definição, linguagens livres de contexto determinísticas são aquelas que podem ser reconhecidas por autômatos de pilha determinísticos;
- Em outras palavras, uma linguagem livre de contexto é dita determinística se for possível demonstrar a existência de pelo menos um autômato de pilha determinístico que a reconheça.

Linguagem livre de contexto não-determinística

A linguagem para a qual se pode provar a inexistência de quaisquer autômatos de pilha determinísticos que a reconheçam, e, simultaneamente, a existência de pelo menos um autômato de pilha não-determinístico que a reconheça, é denominada **linguagem livre de contexto não-determinística**, e o conjunto de tais linguagens constitui uma classe importante das linguagens livres de contexto gerais.

$LR(k)$ e $LL(k)$

A classe das linguagens $LR(k)$ e a classe das linguagens $LL(k)$ situam-se entre a classe das linguagens regulares e a classe das linguagens livres de contexto genéricas, conforme ilustrado na Figura 25.

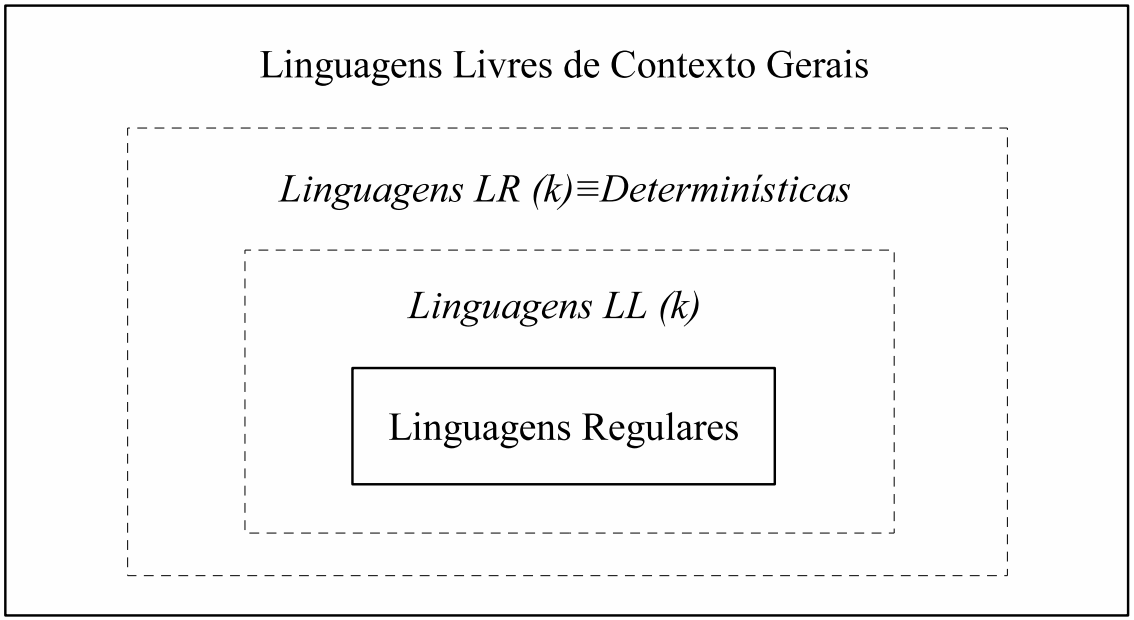

Figura 25: As subclasses $LL(k)$ e $LR(k)$

Demonstrações

- A demonstração da equivalência entre a classe das linguagens livres de contexto que podem ser reconhecidas deterministicamente e a classe das linguagens $LR(k)$ pode ser encontrada em Hopcroft69;
- A demonstração da existência de um algoritmo para determinar se uma dada gramática é $LR(k)$ para um certo valor de k também pode ser encontrada em Hopcroft69.

Linguagens livres de contexto $\times$ linguagens de programação

- As linguagens e as gramáticas livres de contexto determinísticas, sejam elas $LR(k)$ ou $LL(k)$, costumam ser empregadas na formalização sintática da maioria das linguagens de programação convencionais;
- Como conseqüência, os compiladores dessas linguagens costumam ser construídos com base nos correspondentes autômatos de pilha determinísticos;
- Tais reconhecedores, que servem como ponto de partida para a construção de analisadores sintáticos, são com freqüência usados como núcleos dos compiladores das respectivas linguagens, e a eles são agregados os demais módulos de análise e de geração de código, para completar as funções do compilador.

Principais resultados

- As linguagens livres de contexto são fechadas em relação às operações de união, concatenação e fecho de Kleene;
- Elas não são fechadas em relação às operações de complemento e intersecção;
- Os teoremas seguintes apresentam as respectivas demonstrações.

União

Teorema 13.1 "As linguagens livres de contexto são fechadas em relação à operação de união."

Sejam L_1 e L_2 duas linguagens livres de contexto quaisquer. Então, $L_1 = L_1(G_1)$ e $L_2 = L_2(G_2)$, G_1 e G_2 sendo duas gramáticas livres de contexto: $G_1 = (V_1, \Sigma_1, P_1, S_1)$ e $G_2 = (V_2, \Sigma_2, P_2, S_2)$. Admita-se que $N_1 \cap N_2 = \emptyset$. Se isso não for verdade, os símbolos não-terminais podem ser facilmente renomeados, sem prejuízo para as linguagens que estiverem sendo definidas.

União

A linguagem $L_3 = L_1 \cup L_2$ é gerada pela gramática G_3, ou seja, $L_3 = L_3(G_3)$, com $G_3 = (V_3, \Sigma_3, P_3, S_3)$, onde:

- $V_3 = V_1 \cup V_2 \cup \{S_3\}$
- $\Sigma_3 = \Sigma_1 \cup \Sigma_2$
- $P_3 = P_1 \cup P_2 \cup \{S_3 \to S_1, S_3 \to S_2\}$

Como, por hipótese e por construção, as regras contidas em P_3 atendem, todas, às condições necessárias para uma gramática ser classificada como livre de contexto, segue que G_3 é livre de contexto e, conseqüentemente, L_3 é uma linguagem livre de contexto.

Concatenação

Teorema 13.2 "As linguagens livres de contexto são fechadas em relação à operação de concatenação."

Sejam L_1 e L_2 duas linguagens livres de contexto quaisquer. Então, $L_1 = L_1(G_1)$ e $L_2 = L_2(G_2)$, G_1 e G_2 sendo duas gramáticas livres de contexto: $G_1 = (V_1, \Sigma_1, P_1, S_1)$ e $G_2 = (V_2, \Sigma_2, P_2, S_2)$. Admita-se que $N_1 \cap N_2 = \emptyset$. Se isso não for verdade, os símbolos não-terminais podem ser facilmente renomeados, sem prejuízo para as linguagens que estiverem sendo definidas.

Concatenação

A linguagem $L_3 = L_1 L_2$ é gerada pela gramática G_3, ou seja, $L_3 = L_3(G_3)$, com $G_3 = (V_3, \Sigma_3, P_3, S_3)$, onde:

- $V_3 = V_1 \cup V_2 \cup \{S_3\}$
- $\Sigma_3 = \Sigma_1 \cup \Sigma_2$
- $P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_1 S_2\}$

Como, por hipótese e por construção, as regras contidas em P_3 atendem, todas, às condições necessárias para uma gramática ser classificada como livre de contexto, segue que G_3 é livre de contexto e, conseqüentemente, L_3 é uma linguagem livre de contexto.

Fechamento

Teorema 13.3 "As linguagens livres de contexto são fechadas em relação à operação de fecho de Kleene."

Seja L_1 uma linguagem livre de contexto qualquer. Então, $L_1 = L_1(G_1)$, G_1 sendo uma gramática livre de contexto: $G_1 = (V_1, \Sigma_1, P_1, S_1)$.

Fechamento

A linguagem $L_2 = L_1^*$ é gerada pela gramática G_2, ou seja, $L_2 = L_2(G_2)$, com $G_2 = (V_2, \Sigma_2, P_2, S_2)$, onde:

- $V_2 = V_1 \cup \{S_2\}$
- $\Sigma_2 = \Sigma_1$
- $P_2 = P_1 \cup \{S_2 \to S_1 S_2, S_2 \to \varepsilon\}$

Como, por hipótese e por construção, as regras contidas em P_2 atendem, todas, às condições necessárias para uma gramática ser classificada como livre de contexto, segue que G_2 é livre de contexto e, conseqüentemente, L_2 é uma linguagem livre de contexto.

Intersecção

Teorema 13.4 "As linguagens livres de contexto não são fechadas em relação à operação de intersecção."

É suficiente, para provar este teorema, que existem pelo menos duas linguagens livres de contexto cuja intersecção gera uma linguagem que não seja livre de contexto.
Considerem-se as linguagens $L_1 = \{a^m b^m c^n \mid m \geqslant 1, n \geqslant 1\}$ e $L_2 = \{a^n b^m c^m \mid m \geqslant 1, n \geqslant 1\}$. É fácil demonstrar que essas linguagens são geradas, respectivamente, pelas gramáticas livres de contexto G_1 e G_2:

$$\begin{aligned} G_1 &= (\{a,b,c,S,X,Y\},\{a,b,c\}, \\ &\quad \{S \to XY, X \to aXb, X \to ab, Y \to cY, Y \to c\}, S) \\ G_2 &= (\{a,b,c,S,X,Y\},\{a,b,c\}, \\ &\quad \{S \to XY, X \to aX, X \to a, Y \to bYc, Y \to bc\}, S) \end{aligned}$$

Intersecção

Por outro lado, a linguagem $L_3 = L_1 \cap L_2 = \{a^m b^m c^m \mid m \geqslant 1\}$ é uma linguagem que não é livre de contexto, conforme ilustrado na aplicação do “Pumping Lemma” para linguagens livres de contexto (ver Exemplo 11.2).

Intersecção com linguagem regular

Não obstante este resultado, é possível demonstrar que a intersecção de uma linguagem livre de contexto com uma linguagem regular gera sempre uma linguagem livre de contexto.

Complementação

Teorema 13.5 "As linguagens livres de contexto não são fechadas em relação à operação de complementação." Dadas duas linguagens L_1 e L_2 quaisquer, é sabido (Lei de De Morgan) que:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

Nesta igualdade estão envolvidas apenas as operações de intersecção, união e complementação. Sabe-se, conforme demonstração anterior (Teorema 13.1), que as linguagens livres de contexto são fechadas em relação à operação de união. Se elas fossem fechadas também em relação à operação de complemento, deveriam necessariamente ser fechadas em relação à operação de intersecção. Como isso não é verdade (Teorema 13.4), conclui-se que as linguagens livres de contexto não são fechadas em relação à operação de complementação.

Introdução

- Assim como acontece com a classe das linguagens regulares, existem diversas questões acerca das linguagens livres de contexto que podem sempre ser decididas, quaisquer que sejam as linguagens envolvidas;
- Por outro lado, certas questões que são decidíveis para a classe das linguagens regulares não podem ser decididas no caso geral, quando transpostas para a classe das linguagens livres de contexto (por exemplo, a questão $L = \Sigma^*$ é decidível no caso das linguagens regulares, porém não é decidível no caso das linguagens livres de contexto).

Pertencimento

Teorema 14.1 "Sejam L uma linguagem livre de contexto sobre Σ e α uma cadeia pertencente a Σ^*. Então, a questão "$\alpha \in L$?" é decidível."

Conforme o Algoritmo 14.1.

Pertencimento

Algoritmo 14.1 *"Determinação da pertinência da cadeia $w \in \Sigma^*$ à linguagem livre de contexto $L \subseteq \Sigma^*$."*

- *Entrada: uma cadeia $w \in \Sigma^*$ e uma linguagem livre de contexto $L \subseteq \Sigma^*$;*
- *Saída: SIM, se $w \in \Sigma^*$; NÃO, caso contrário;*

Pertencimento

Método:

- *Obter uma gramática livre de contexto G tal que $L = L(G)$;*
- *Se $w = \varepsilon$, então:*
  1. *Determinar, conforme o Algoritmo 5.3 (eliminação de produções em vazio em gramáticas livres de contexto), se S (a raiz de G) pertence ao conjunto E; em caso afirmativo, a resposta é SIM; caso contrário, a resposta é NÃO;*

Pertencimento

- *Se $w \neq \varepsilon$, então:*
  1. *Obter G' na Forma Normal de Greibach, tal que $L(G') = L(G) - \{\varepsilon\}$;*
  2. *Considerar $n = |w|$;*
  3. *Considerar m como o maior número de produções definido para um não-terminal, entre todos os não-terminais de G';*
  4. *Obter todas as seqüências de derivações mais à esquerda que geram formas sentenciais cujo prefixo seja uma cadeia de terminais de comprimento máximo n (existem no máximo m^n seqüências distintas);*
  5. *Verificar se alguma dessas seqüências de derivação corresponde à geração da cadeia w; em caso afirmativo, a resposta é SIM; caso contrário, a resposta é NÃO.*

Exemplo

Exemplo 14.1
Considere-se a gramática abaixo, já apresentada na Forma Normal de Greibach:

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

Considere-se a cadeia $abbc \in L(G)$. Então, $n = 4$ e $m = 2$ (pois existem duas produções para S, duas para B e apenas uma para C) e, conforme o Algoritmo 14.1, existem no máximo $2^4 = 16$ seqüências distintas de derivações mais à esquerda que geram como prefixo uma cadeia de terminais de comprimento máximo 4, não havendo necessidade de se inspecionar outras seqüências. A Tabela 1 relaciona todas as oito seqüências que geram cadeias de terminais de comprimento máximo 4.

Exemplo

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

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

Exemplo

Como se pode verificar, a seqüência de derivações 3 produz a cadeia *abbc*. Por outro lado, a cadeia *bcbc*, também de comprimento 4, não pertence à linguagem, uma vez que nenhuma das oito seqüências da Tabela 1 gera o prefixo *bcbc* e, portanto, nenhuma seqüência de derivações é capaz de gerar a cadeia *bcbc*.

Tempo de execução

- O tempo de execução do Algoritmo 14.1 é proporcional a m^n, onde n é o comprimento da cadeia de entrada;
- Logo, esse tempo varia exponencialmente com o tamanho da cadeia, o que é um resultado considerado ineficiente, e portanto indesejável do ponto de vista prático;
- Diversos outros algoritmos, no entanto, apresentam tempos de execução que são proporcionais a n^3 (ou menos), o que implica importantes ganhos de desempenho;
- Uma discussão sobre tais algoritmos, assim como referências, pode ser encontrada em Hopcroft79, ou na literatura sobre compiladores, e foge ao escopo da presente publicação.

Pertencimento

- Fica, portanto, demonstrada a existência de autômatos de pilha que sempre param, quaisquer que sejam a linguagem livre de contexto considerada e a cadeia de entrada que lhes sejam submetidas;
- Essa propriedade é análoga à anteriormente (pertinência de uma cadeia a uma linguagem regular), que garante a existência de autômatos finitos que sempre param, quaisquer que sejam a linguagem regular considerada e a cadeia de entrada que lhes seja submetida.

Vazia?

Teorema 14.2 "Seja L uma linguagem livre de contexto. Então, a questão "$L = \emptyset$?" é decidível." $L = \emptyset$ se e somente se L não contém nenhuma sentença de comprimento menor que n, onde n é a constante definida pelo "Pumping Lemma" para linguagens livres de contexto. Tal fato pode ser verificado por demonstração semelhante à que foi feita anteriormente (determina se uma linguagem regular é vazia ou não-vazia).
Uma maneira alternativa de se verificar se $L = \emptyset$ consiste em obter uma gramática $G = (V, \Sigma, P, S)$, sem símbolos inúteis, que gere L. Se $S \in V$, então L é não-vazia. Caso contrário, L é vazia.

Infinita?

Teorema 14.3 "Seja L uma linguagem livre de contexto. Então, a questão "L é infinita?" é decidível." L é infinita se e somente se contiver pelo menos uma sentença de comprimento maior ou igual a n e menor que $2n$, onde n é a constante definida pelo "Pumping Lemma" para linguagens livres de contexto. Tal fato pode ser verificado por demonstração semelhante à que foi feita anteriormente (determina se uma linguagem regular é finita ou infinita).
Uma outra maneira de se determinar se L é infinita consiste em obter uma gramática sem símbolos inúteis que gere L, e depois verificar se existem não-terminais X auto-recursivos na mesma $(X \Rightarrow \alpha X \beta, \alpha\beta \in \Sigma^+)$. Caso exista pelo menos um não-terminal auto-recursivo (não necessariamente central), então L é infinita. Caso contrário, L é finita.

Questões não-decidíveis

As seguintes questões acerca das linguagens e das gramáticas livres de contexto (respectivamente L, G, G_1 e G_2 quaisquer) não são decidíveis e não serão demonstradas neste livro.

- G é ambígua?
- L é inerentemente ambígua?
- $L(G)$ é regular?
- $L(G) = \Sigma^*$?
- $L(G_1) = L(G_2)$?
- $L(G_1) \subseteq L(G_2)$?
- $L(G_1) \cap L(G_2) = \emptyset$?
- $L(G_1) \cap L(G_2)$ é livre de contexto?
- $\overline{L(G_1)}$ é livre de contexto?