Unidade II

5 LINGUAGENS LIVRES DE CONTEXTO — PARTE 1

As Linguagens Livres de Contexto são geradas por Gramáticas Livres de Contexto.

A Gramática Livre de Contexto foi definida por Chomsky (1956). Trata-se de um dispositivo mais poderoso que aquele das Linguagens Regulares, visto que apresenta regras que permitem expressar como os elementos de uma forma sentencial se organizam em **grupos hierárquicos** complexos.

De fato, considere o seguinte exemplo:

A limpava menina casa a vassoura uma com.

Trata-se de uma sentença não gramatical, dado que os seus elementos constituintes não estão agrupados corretamente.

A organização adequada dos elementos da frase é apresentada no quadro a seguir.

<table>
<tr><td colspan="8">a menina limpava a casa com uma vassoura</td></tr>
<tr><td colspan="2">a menina</td><td colspan="6">limpava a casa com uma vassoura</td></tr>
<tr><td>a</td><td>menina</td><td>limpava</td><td colspan="2">a casa</td><td colspan="3">com uma vassoura</td></tr>
<tr><td colspan="3"></td><td>a</td><td>casa</td><td>com</td><td colspan="2">uma vassoura</td></tr>
<tr><td colspan="6"></td><td>uma</td><td>vassoura</td></tr>
</table>

Figura 29

As Gramáticas Livres de Contexto representam um importante formalismo para o processamento das linguagens naturais, bem como das artificiais, em particular das linguagens de programação.

O estudo das Linguagens Livres de Contexto é aplicado no projeto e na implementação do analisador sintático, módulo funcional do compilador.

Exemplo: apresenta-se uma Gramática Livre de Contexto para uma pequena linguagem de programação (SEBESTA, 2003).

G = (V, Σ, P, <programa>), onde:

V = {<programa>, <lista_inst>, <inst>, <comando>, <var>, <expressão>}

Σ = {A, B, C, +, -, begin, end}

<programa> é o símbolo inicial

P = {<programa> → begin <lista_inst> end

<lista_inst> → <inst> | <commando>; <lista_inst>

<inst> → <var> = <expressão>

<var> → A | B | C

<expressão> → <var> + <var> | <var> - <var> | <var>

5.1 Gramática Livre de Contexto: dispositivo gerador de uma Linguagem Livre de Contexto

Seja G = (V, Σ, P, S). Diz-se que G é livre de contexto, se as produções apresentarem o seguinte padrão:

$$A \rightarrow \beta, \text{ onde: } A \in V \text{ e } \beta \in (V \cup T)^*$$

A expressão acima indica que no lado esquerdo da produção deve existir um e, apenas um, símbolo não terminal e, no lado direito da produção, podem figurar quaisquer cadeias de símbolos, sejam terminais, não terminais e até mesmo isoladamente, a cadeia vazia.

Uma linguagem L definida sobre um alfabeto Σ é livre de contexto se pode ser gerada por uma Gramática Livre de Contexto.

A seguir são apresentados alguns exemplos de Gramáticas Livres de Contexto. Em cada um destes exemplos observe o formato das produções: no lado esquerdo da produção figura um e, apenas um, símbolo não terminal e, no lado direito, quaisquer sequências de símbolos terminais e não terminais.

Exemplo: É conhecido o duplo balanceamento dos símbolos "(" e ")", nas expressões lógico-aritméticas das linguagens de programação, bem como o duplo balanceamento dos símbolos "{" e "}" na estruturação de blocos de comandos. Considere a seguinte linguagem L com duplo balanceamento e definida a partir do alfabeto Σ:

$\Sigma = \{a, b\}$ e $L = \{\omega \mid \omega = a^n b^n, n \geq 0\}$

A gramática G, geradora, é:

G = (V,Σ, P, S) = ({S}, {a,b}, P, S)

P = {S → aSb | ε}

Confira:

$S \Rightarrow \varepsilon$

$S \Rightarrow aSb \Rightarrow a\varepsilon b = ab$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow = aa\varepsilon bb = aabb$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaa\varepsilon bbb = aaabbb$

Exemplo: Considere a gramática G = (V, Σ, P, S), tal que:

V = {S, A, O, Y}

Σ = {x, y, z, +, *, -, /}

P = {S → (AOA)

A → Y | S

O → + | - | * | /

Y→ a|b|c}

Tem-se que:

S ⇒(AOA) ⇒(YOA) ⇒(aOA) ⇒(a*A) ⇒(a*S) ⇒(a*(AOA)) ⇒(a*(YOA)) ⇒ (a*(b OA)) ⇒(a*(b OA)) ⇒(a*(b /A)) ⇒(a*(b /Y)) ⇒(a*(b /c))

Observe que há linguagens livres de contexto que não são regulares. Por outro lado, todas as linguagens regulares são livres de contexto.

Exemplo: As regras 1 a 6 pertencem ao conjunto de produções P de uma gramática G = ({E, T, F}, {id, , +, *, (,) }, P, E).

1: E → E + T

2: E → T

3: T → T * F

4: T → F

5: F → (E)

6: F → id

Pede-se:

a) Mostre que G gera a cadeia (id + id * id) * (id + id)

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

$(F * F + T) * F \Rightarrow (id * id + T) * F \Rightarrow (id * id + F) * F \Rightarrow (id * id + id) * F \Rightarrow$

$(id* id + id) * (E) \Rightarrow (id * id + id) * (E + T) \Rightarrow (id *id + id) * (T + T) \Rightarrow$

$\Rightarrow (id *id + id) * (F + T) \Rightarrow (id *id + id) * (id + T) \Rightarrow (id *id + id) * (id + F) \Rightarrow$

$\Rightarrow (id *id + id) * (id + id)$

b) Por que essa gramática é classificada como livre de contexto?

Porque as produções são da forma: $A \rightarrow \alpha$, onde $A \in V$ e $\alpha \in (V \cup T)^*$

V = conjunto dos símbolos não terminais e T = conjunto dos símbolos terminais.

Árvores de derivação

Uma **árvore de derivação** é uma representação gráfica de uma derivação que filtra a ordem na qual as produções são aplicadas para substituir não terminais.

Formalmente, uma árvore de derivação é um sistema de representação de sequências de derivações em que uma gramática livre de contexto G = (V, Σ, P, S) consiste em um grafo orientado e ordenado, acíclico, com as seguintes propriedades:

1. Todo nó é rotulado com um elemento de $(V \cup \Sigma \cup \varepsilon)$.
2. O rótulo da raiz é S, o símbolo inicial da gramática.
3. Os rótulos dos nós internos são elementos de V, o conjunto dos símbolos não terminais da gramática.
4. Se um nó tem rótulo A, e A_1, A_2, ...,A_n são descendentes diretos de A, ordenados da esquerda para a direita, então existe uma produção da gramática da forma $A \rightarrow A_1A_2....A_n$.
5. Se um nó possui um rótulo ε, então este nó deve ser simultaneamente uma folha e descendente único de seu ancestral direto.

Exemplo: Seja a gramática G = ({S, B}, {a, b}, P, S), onde:

P= {S → aB | aSB; B→ b}.

A derivação para a cadeia aabb é:

$S \Rightarrow aSB \Rightarrow aaBB \Rightarrow aabB \Rightarrow aabb$

A árvore de derivação dessa cadeia é representada na figura a seguir:

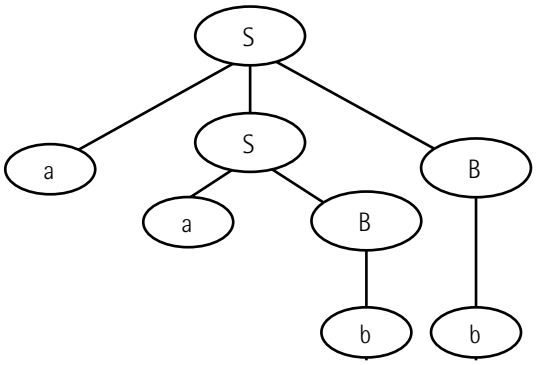

Figura 30 – Exemplo de árvore de derivação

Gramáticas ambíguas

Uma gramática livre de contexto é ambígua se existir uma palavra que possua duas ou mais árvores de derivação.

Exemplo: Considere-se a seguinte gramática:

$$G = (\{E\}, \{x, +, *\}, P, E), \text{ onde:}$$

$$P = \{E \rightarrow E + E \mid E * E \mid x\}.$$

A cadeia x + x * x é gerada pela gramática G.

Tem-se que:

$E \Rightarrow E + E \Rightarrow E + E \Rightarrow x + E \Rightarrow x + E * E \Rightarrow x + x * E \Rightarrow x + x * x.$

A árvore de derivação correspondente é:

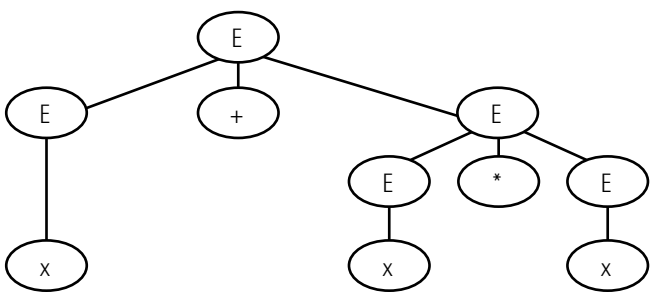

Figura 31 – Árvore de derivação para a sentença x+x*x

Por outro lado, a mesma sentença, x+ x * x pode ser derivada como se segue:

$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow x + E * E \Rightarrow x + x * E \Rightarrow x + x * E \Rightarrow x + x * x.$

A árvore de derivação correspondente é:

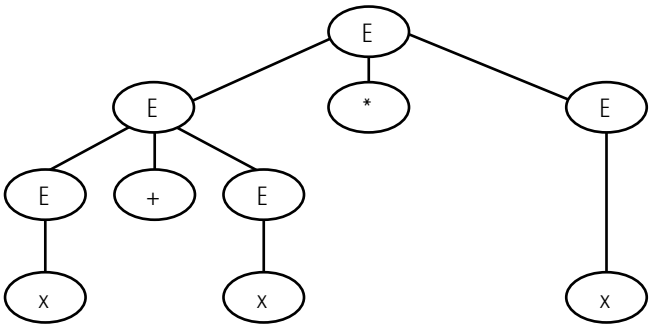

Figura 32 – Árvore de derivação para a sentença x+x*x

Exemplo: Mostre que a seguinte gramática G = (V, T, P, S) é ambígua.

$V = \{S\}$

$T = \{(,)\}$

$P = \{S \rightarrow SS \mid \varepsilon \mid (S)\ \}$

De fato, a palavra ( ) apresenta as seguintes derivações distintas:

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

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

Exemplo: Considere a seguinte gramática livre de contexto:

G =({S}, {a, b}, P, S), onde:

$P = \{S \rightarrow SS \mid aSa \mid bSb \mid \varepsilon\ \}$

Essa gramática é ambígua porque existe pelo menos uma palavra que apresenta mais de uma árvore de derivação possível. Verifique, a título de exercício, as árvores de derivação para as palavras aaaa e bbbb.

Uma linguagem é uma **linguagem inerentemente ambígua** se qualquer Gramática Livre de Contexto que a define é ambígua.

6 AUTÔMATOS DE PILHA: DISPOSITIVOS RECONHECEDORES DE LINGUAGENS LIVRES DE CONTEXTO

As linguagens que são livres de contexto no seu sentido estrito não podem ser reconhecidas por um autômato finito.

Considere-se, como exemplo, a linguagem L definida sobre o alfabeto $\Sigma = \{a, b\}$, de forma que:

$$L = \{\omega \mid \omega \text{ é um palíndromo}\}$$

Um dispositivo reconhecedor dessa linguagem deveria inicialmente ler a primeira metade da cadeia, armazená-la em uma pilha, de forma que, ao processar a segunda metade da cadeia, pudesse compará-la, símbolo a símbolo, com a cadeia recém-armazenada. Naturalmente, o autômato finito não poderia processar essa linguagem, pois se trata de um dispositivo sem memória auxiliar organizada em pilha.

O autômato finito também não pode processar as linguagens com duplo balanceamento. Seja a linguagem $L = \{a^n b^n \mid n \geq 1\}$. Os autômatos finitos não têm como identificar o número de ocorrências do símbolo **a** na cadeia de entrada e comparar com o número de ocorrências do símbolo **b**.

O dispositivo reconhecedor das linguagens livres de contexto é o autômato de pilha **não determinístico**. Cumpre observar que os autômatos finitos **determinísticos** são capazes de reconhecer apenas um subconjunto das linguagens livres de contexto.

Um autômato com pilha **não determinístico** apresenta os seguintes componentes:

- Fita de entrada

Análoga àquela do autômato finito, sua função é armazenar a cadeia a ser analisada. A fita de entrada é também dividida em células, cujo número é igual ao número de símbolos da cadeia de entrada. Dispõe de um cursor, que se movimenta apenas da esquerda para a direita. Os símbolos presentes na fita de entrada não podem ser alterados, ou seja, a operação que se efetua sobre a mesma é apenas de leitura, e não de gravação.

- Unidade de controle

Possui um número finito e predefinido de estados. As transições de estados do autômato de pilha se fazem mediante as especificações previstas por uma **relação** de transição. Essa relação associa o estado corrente do autômato a um novo estado. Essa mudança de estado é especificada a partir da leitura do símbolo corrente da cadeia de entrada, ou pode ser em vazio; pode ser determinada pela leitura e remoção de um símbolo no topo da pilha e geralmente implica a inserção de um novo símbolo na pilha. Em outras palavras, os símbolos a serem lidos da cadeia de entrada, a serem lidos do topo da pilha e a serem gravados nessa estrutura de dados, podem coincidir com a cadeia vazia, ε.

- Memória auxiliar organizada em pilha:

Os símbolos a serem consultados, removidos e armazenados na pilha, em geral, não coincidem com os símbolos do alfabeto de entrada. Os símbolos da pilha apresentam um alfabeto próprio.

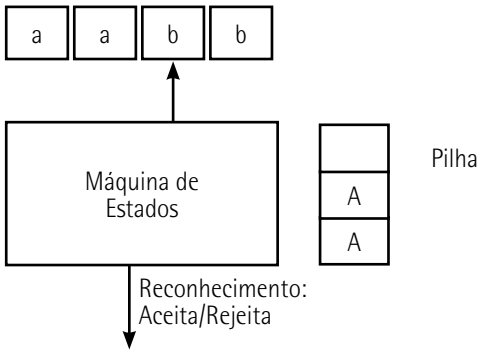

Figura 33 – Autômato de pilha. Conteúdo da pilha após a leituras da subcadeia aa

A partir da ilustração anterior é possível inferir que, uma vez lidos os dois primeiros símbolos **a**", foram inseridos dois símbolos **A** na pilha.

Considere a figura seguinte:

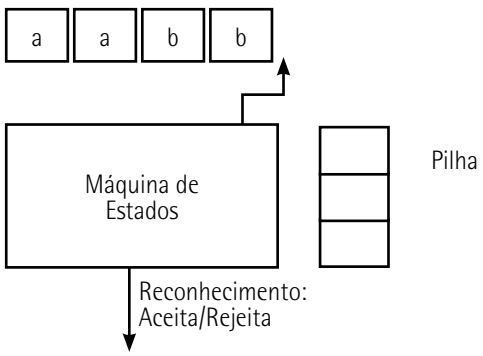

Figura 34 – Autômato de pilha: conteúdo da pilha após as leituras da subcadeia aabb

Pode-se inferir que, após a leitura dos dois símbolos **b**, foram removidos os símbolos **A**, recém-inseridos na pilha. Através da inserção e remoção de elementos na pilha, é possível constar que a cadeia analisada apresenta de fato o balanceamento entre os símbolos **a** e **b**.

A figura a seguir apresenta uma transição entre dois estados de um autômato de pilha. Observe que as transições são rotuladas por uma tripla. O primeiro elemento da tripla é o elemento a ser lido da cadeia de entrada. O segundo elemento coincide com o símbolo a ser consultado e removido do topo da pilha e finalmente o terceiro elemento coincide com o símbolo a ser inserido na pilha após a transição.

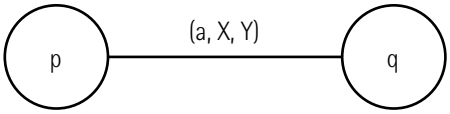

Figura 35 - Transição de um autômato de pilha

Representação algébrica do autômato de pilha

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

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

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;

g é uma **relação** de transição g: $(Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*) \times (Q \times \Gamma^*)$;

q_0 é o estado inicial;

$F \subseteq Q$ é o conjunto de estados finais.

Os dispositivos que fazem uso de pilha podem operar segundo 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 **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 q0, o cursor se encontra posicionado sob a célula mais à esquerda da fita de entrada e o conteúdo da pilha é Z_0.

Exemplo: Considere o autômato de pilha $M = (Q, \Sigma, \Gamma, g, q_0, F)$, tal que:

Conjunto finito de estados $Q = \{q0, qf\}$

Conjunto de estados finais F = {qf}

Alfabeto (finito e não vazio) de entrada Σ = {a, b}

O alfabeto de pilha é Γ = {a}

A função de transição é dada por:

g(q0, a, ε) = (q0, a)

g(q0, b, ε) = (q0,a)

g(q0, a, ε) = (qf, ε)

g(qf, a, a) = (qf, ε)

g(qf, b, a) = (qf, ε)

A representação gráfica do autômato de pilha M é apresentada a seguir:

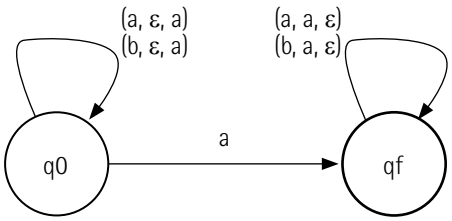

Figura 36 – Exemplo de um autômato de pilha

O reconhecimento da cadeia de entrada aab pode ser descrito como na figura seguinte:

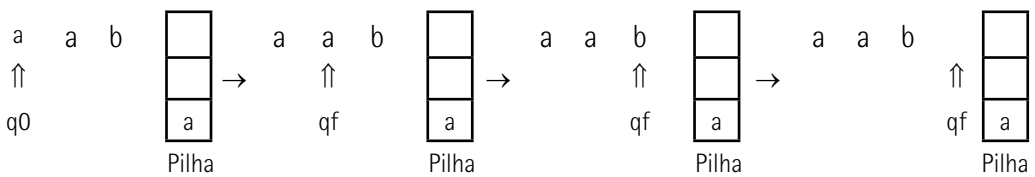

Figura 37

Exemplo: Considere a seguinte gramática G = (V, Σ, P, S), com:

Σ = {a, b, c}

V = {S}

P = {S →a S a | b S b | c)

A linguagem gerada pela gramática G é $L(G) = \{x = \omega c \omega^R \mid x \in \{a, b\}^*\}$

O autômato que aceita essa linguagem é:

$M = (Q, \Sigma, \Gamma, g, p, \{q\})$, tal que:

conjunto finito de estados $Q = \{p, q\}$;

alfabeto (finito e não vazio) de entrada $\Sigma = \{a, b, c\}$;

o alfabeto de pilha é $\Gamma = \{a, S, b\}$;

o estado inicial é p;

$F \subseteq Q$ é o conjunto de estados finais, com $F = \{q\}$.

A relação de transição g é dada por:

$g = \{((p, \varepsilon, \varepsilon), (q, S))\}$,

$((q, \varepsilon, S), (q, aSa))$,

$((q, \varepsilon, S), (q, bSb))$,

$((q, \varepsilon, S), (q,c))$,

$((q, a, a), (q, \varepsilon))$,

$((q, b, b), (q,\varepsilon))$,

$((q, c, c), (q,\varepsilon))\}$

A seguir, ilustra-se a sequência de movimentos do autômato, de sua configuração inicial até sua configuração final, para a cadeia *abbcbba*, através da tabela a seguir:

Tabela 10

Estado corrente	Símbolo corrente na cadeia de entrada (em negrito)	Topo da pilha (em negrito)	Transição	Próximo estado
p	**a**bbcbba	e	((p, ε, ε), (q, **S**))	q
q	**a**bbcbba$	**S**	((q,ε,S), (q,aSa))	q
q	**a**bbcbba	**a**Sa	((q,a,a), (q,ε))	q
q	**b**bcbba	**S**a	((q,ε,S), (q,bSb))	q

q	**b**bcbba	**b**Sba	((q,b,b), (q,ε))	q
q	**b**cbba	**S**ba	((q,ε,S), (q,bSb))	q
q	**b**cbba	**b**Sbba	((q,b,b), (q,ε))	q
q	**c**bba	**S**bba	((q,ε,S), (q,c))	q
q	**c**bba	**c**bba	((q,c,c), (q,ε))	q
q	**b**ba	**b**ba	((q,b,b), (q,ε))	q
q	**b**a	**b**a	((q,b,b), (q,ε))	q
q	**a**	**a**	((q,a,a), (q,ε))	q
q	ε	ε		q

Teoremas sobre as Linguagens Livres de Contexto

Teorema: se L é uma Linguagem Livre do Contexto, então existe M, autômato com pilha que aceita L(M).

Corolário: autômato com pilha × número de estados.

Se L é uma Linguagem Livre do Contexto, então:

a) existe M, autômato com pilha, com controle de aceitação por estados finais, com somente três estados, tal que L = L(M).

b) existe M, autômato com pilha, com controle de aceitação por pilha vazia, com somente um estado, tal que L = L(M).

Teorema: para qualquer Linguagem Livre de Contexto, existe um autômato com pilha que sempre para, qualquer que seja a entrada.

Teorema: se L é aceita por um autômato de pilha, então L é Linguagem Livre de Contexto.

6.1 O lema do bombeamento para Linguagens Livres de Contexto

Assim, como para as Linguagens Regulares, enuncia-se o lema do bombeamento para as Linguagens Livres de Contexto, o qual permite que as propriedades das mesmas sejam analisadas.

Lema do Bombeamento

Se L é uma Linguagem Livre de Contexto, então existe uma constante n tal que, para qualquer palavra w de L, onde $|w| \geq n$, w pode ser definida como w = uxvyz, onde $|xvy| \leq n$, $|xy| \geq 1$ e, para todo $i \geq 0$, u x^i v y^iz é palavra de L.

Exemplo de aplicação

Questão 1. O conjunto das Linguagens Livres de Contexto é fechado para a operação de união, mas não o é para as operações de intersecção ou complemento. Verifique se as seguintes linguagens são Livres de Contexto.

$L(w) = \{ w = a^m b^n \mid m \neq n \}$

$L(w) = \{a, b\}^* - \{a^n b^n \mid n \geq 0 \}$

$L(w) = \{a^n b^n c^n \mid n \geq 0 \}$

Questão 2. Para cada uma das linguagens a seguir, construa um autômato de pilha.

$L = \{w \mid w = a^n b^n c^m, n > 0, m>0\}$

$L = \{w \mid w = a^n b^m c^n d^l, n > 0, m> 0, l >0\}$

$L = \{w \mid w = a^n b^n c^m d^m, n > 0, m > 0\}$

7 ALGORITMOS DE ANÁLISE SINTÁTICA

Um analisador sintático de uma linguagem gerada por uma gramática verifica se é possível construir uma árvore de derivação para uma dada cadeia de entrada.

Segundo Aho *et al.* (2008), existem três estratégias gerais de análise sintática para o processamento de gramáticas: universal, descendente e ascendente.

Os métodos de análise universal podem analisar qualquer gramática, no entanto, são muito ineficientes. São exemplos desses algoritmos o algoritmo de Cocke-Younger-Kasami e o algoritmo de Early.

Os compiladores, em geral, empregam, em vez dos métodos universais, aqueles conhecidos como determinísticos ascendentes e determinísticos descendentes.

Os analisadores determinísticos descendentes constroem a árvore de derivação para a palavra de entrada (a ser reconhecida) a partir da raiz (símbolo inicial da gramática), gerando os ramos em direção às folhas (símbolos terminais que compõem a palavra).

Naturalmente, os analisadores ascendentes constroem a árvore de derivação das folhas para a raiz.

Os métodos ascendentes e descendentes mais eficientes funcionam apenas para subclasses de gramáticas, e ambos analisam a cadeia de entrada da esquerda para a direita.

A seguir são apresentados os algoritmos de Cocke-Younger-Kasami, bem como são tecidas considerações sobre algoritmos determinísticos ascendentes.

Algoritmo de Cocke Younger Kasumi

O algoritmo de Cocke-Younger-Kasami foi proposto em 1965. Trata-se de um algoritmo ascendente e emprega a **Forma Normal de Chomsky**.

Uma gramática Livre de Contexto é dita na **Forma Normal de Chomsky** se todas as suas produções são da forma:

$A \rightarrow BC$ ou $A \rightarrow a$

onde: $A, B, C \in V$ e $a \in \Sigma$.

Seja $G = (V, \Sigma, P, S)$ uma gramática na **Forma Normal de Chomsky**, onde $\Sigma = \{a_1, a_2, \ldots a_n\}$, e suponha $\omega = a_1 a_2 \ldots a_n$ uma entrada a ser verificada. O algoritmo emprega uma tabela triangular, conforme ilustra a figura seguinte:

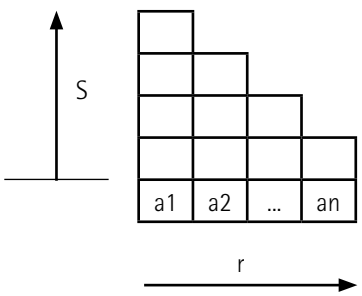

Figura 38 - Tabela triangular para os algoritmos de Cocke–Younger-Kasami

O algoritmo de Cocke-Younger-Kasami (CYK) é composto pelas etapas apresentadas a seguir. Empregar-se-á o símbolo Xrs para nomear as células da tabela triangular de derivação, a saber: o índice "r" indica a coluna e o índice "s" diz respeito à linha.

a) Inicialmente, investigam-se todas as regras que geram os símbolos terminais, ou seja, regras da forma: $A \rightarrow a$. Para cada regra r que apresenta esse padrão, deve-se preencher cada célula X_{r1} (da primeira linha) com o símbolo não terminal A.

b) A partir da segunda linha até a última linha n, cada célula de cada coluna deve ser preenchida. Lembrando que a tabela é triangular, deve-se variar o contador de linhas s, de 2 a n, e o contador de colunas r de 1 até n-s+1. O preenchimento de cada célula X_{rs} se faz mediante a investigação de todas as regras da forma: $A \rightarrow BC$. O objetivo dessa etapa é o de identificar todos os não terminais que serão armazenados na célula X_{rs} que gerem a subcadeia $a_r a_{r+1} \ldots a_s$. Assim, $A \in X_{rs}$, se existe uma regra $A \rightarrow BC$, tal que B se apresente na mesma coluna r, situado em linhas abaixo da linha corrente s, e o não terminal C se apresente nas colunas à direita da coluna r, nas linhas abaixo da

linha s. Assim sendo, esses não terminais da célula X_{rs} são obtidos através do conjunto de iterações descrito pelo seguinte pseudocódigo:

Xrs = ∅; Para k variando de 1 até s-1, faça:

$$Xrs = Xrs \cup \{A \mid A \rightarrow BC \in P, B \in Xrk \text{ e } C \in X(r+k)(s-k)\}$$

A condição de aceitação da entrada implica o símbolo inicial da gramática (raiz da árvore de derivação de toda palavra) se apresentar na célula X1n. Nessa situação, a entrada é aceita pelo algoritmo.

Exemplo: Seja a gramática G = (V, Σ, P, S), onde:

$V = \{S, A\}$

$\Sigma = \{a, b\}$

$P = \{S \rightarrow AB; S \rightarrow b; A \rightarrow a; B \rightarrow S\ A\}$

Pede-se verificar se a cadeia aabaa é gerada pela gramática G, empregando-se o algoritmo de Cocke-Younger-Kasami.

Inicialmente, constrói-se uma tabela triangular. Para melhor compreensão do algoritmo, apresentam-se as células da tabela triangular nomeadas, conforme se segue:

X15				
X14	X24			
X13	X23	X33		
X12	X22	X32	X42	
X11	X21	X31	X41	X51
a	a	b	a	a

Figura 39 – Tabela triangular com células nomeadas

A primeira etapa do algoritmo diz respeito às produções da forma $A \rightarrow a$ e ao preenchimento das células X_{11}, X_{21}, X_{31}, X_{41} e X_{51}.

O preenchimento das células prossegue na seguinte ordem: X_{12}, X_{22}, X_{32} e X_{42}, sucedidas pelas células X_{13}, X_{23} e X_{33}, X_{14}, X_{24} e, finalmente, X_{15}.

A fim de melhor ilustrar a ordem do preenchimento das células, apresenta-se a tabela a seguir, onde se elencam os valores de r, s e k durante as diversas iterações previstas no algoritmo. Apresentam-se também o conteúdo das células X_{rk}, $X_{r+k, s-k}$ e o conteúdo resultante da célula X_{rs}.

Tabela 11

N = 5 (comprimento da cadeia de entrada)					
s	r	k	X_{rk}	$X_{r+k, s-k}$	X_{rs}
2	1	1	X_{11} =A	X_{21} = A	X_{12} = -
	2	1	X_{21} -	X_{31} = -	X_{22}= -
	3	1	X_{31} = S	X_{41} = A	X_{32}= B
	4	1	X_{41} = A	X_{51} = A	X_{42} = -
3	1	1	X_{11}=A	X_{22}= -	X_{13} = -
		2	X_{12}=-	X_{31} = S	
	2	1	X_{21} = A	X_{32} = B	X_{23} = S
		2	X_{22} = -	X_{41} = A	
	3	1	X_{31} = S	X_{42} = -	X_{33} = -
		2	X_{32} = B	X_{51} = A	
4	1	1	X_{11} = A	X_{23} = S	X_{14} -
		2	X_{12} -	X_{32} = B	
		3	X_{13} = -	X_{41} = A	
	2	1	X_{21} = A	X_{33} = -	X_{24} = B
		2	X_{22} = -	X_{42} = -	
		3	X_{23} = S	X_{51} = A	
5	1	1	X_{11} = A	X_{24}= B	X_{15} = S
		2	X_{12} = -	X_{33} = -	
		3	X_{13} = -	X_{42} -	
		4	X_{14} = -	X_{51} = A	

A seguir, apresenta-se a sequência de preenchimento das células na tabela triangular.

1 - Preenchimento das células: X_{11}, X_{21},...X_{51}.

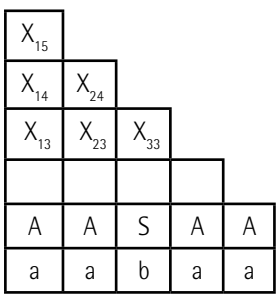

Figura 40

2 – Preenchimento das células X_{12}, X_{22}, X_{32} e X_{42}.

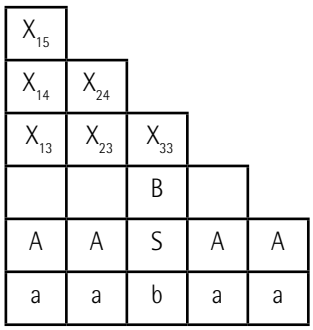

Figura 41

3 - Preenchimento das células X_{13}, X_{23} e X_{33}.

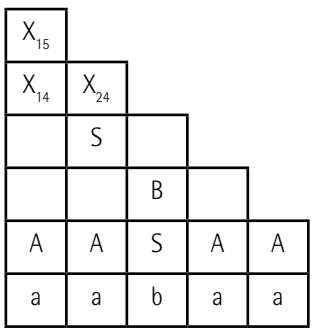

Figura 42

4 - Preenchimento das células X_{14}, X_{24} e X_{15}.

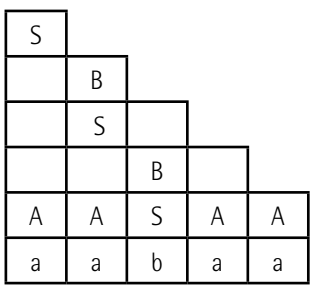

Figura 43

Como o símbolo inicial ocorre na última célula (X_{15}) calculada, infere-se que a cadeia aabaa é gerada pela gramática G.

Analisador sintático ascendente

Os analisadores sintáticos ascendentes analisam a cadeia de entrada da esquerda para a direita e constroem a árvore de derivação das folhas para a raiz. Pode-se pensar na análise ascendente como o processo de "reduzir" uma cadeia ω para o símbolo inicial da gramática.

Os analisadores sintáticos ascendentes podem ser construídos através de diferentes métodos.

Constrói-se um autômato de pilha como se segue:

Seja G = (V, Σ, P, S) uma gramática livre de contexto e seja o autômato de pilha M = (Q, Σ, g, q0, F, Γ), onde Q = {q0, qf}, Γ = V e F = {qf}.

Tem-se que g, a relação de transição, é obtido como se segue:

- g(p, a, ε) = (q0, a), para cada a ∈ ∑. Observe-se que se trata de uma transição de inserção de símbolo na pilha.
- g(p, ε, α^R) = (q0, A), para cada A → α. Trata-se da substituição do lado direito da regra pelo lado esquerdo. Encontra-se na pilha o reverso da forma sentencial α.
- g(p, ε, S) = = (qf, ε). Observe-se que se trata de uma transição com remoção de símbolo da pilha.

Exemplo: Considerem as seguintes regras:

(1) E → E + T

(2) E → T

(3) T → T * F

(4) T → F

(5) F → (E)

(6) F → x

O autômato de pilha pode ser obtido como se segue:

(g0) g(q0, a, ε) = (q0, a), para cada a ∈ ∑.

(g1) g(q0, ε, T+E) = (q0, E)

(g2) g(q0, ε,T) = (q0, E)

(g3) g(q0, ε, F*T) = (q0, T)

(g4) g(q0, ε, F) = (q0, T)

(g5) g(q0, ε,)E( ) = (q0, F)

(g6) g(q0, ε, x) = (q0, F)

(g7) g(q0, ε, E) = (qf, ε)

Pode-se verificar que a cadeia de entrada x*x + x é aceita pelo autômato M projetado:

Tabela 12

Etapa	Estado	Entrada	Pilha	Transição Usada	Regra
0	q0	x*x + x	e		
1	q0	*x+x	x	g0	
2	q0	*x+x	F	g6	6 :F→x
3	q0	*x+x	T	g4	4 :T → F
4	q0	x+x	*T	g0	
5	q0	+x	x*T	g0	
6	q0	+x	F*T	g6	6: F→x
7	q0	+x	T	g3	3: T → T * F
8	q0	+x	E	g2	2: E → T
8	q0	x	+E	g0	
9	q0	e	x+E	g0	
10	q0	e	F + E	g6	6 F→x
11	q0	e	T + E	g4	4 T → F
12	q0	e	E	g1	1 E → E + T
13	qf	e	e	g7	

Observe que a seguinte árvore de derivação foi gerada para a cadeia x*x + x.

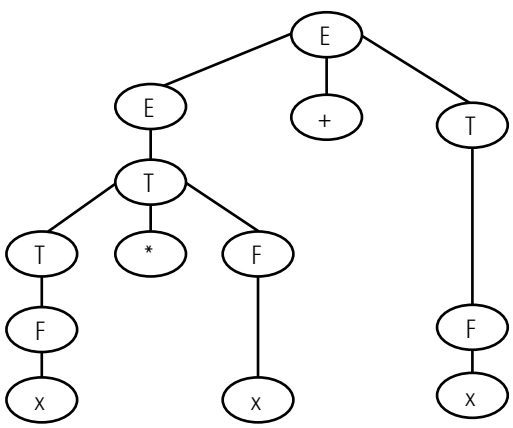

Figura 44 - Árvore de derivação para a sentença x*x+x

É possível perceber que o autômato projetado é não determinístico.

Os compiladores "de produção", ou seja, aqueles presentes nos ambientes de desenvolvimento disponíveis no mercado, devem empregar algoritmos determinísticos.

A seguir, apresenta-se a estrutura de um autômato de análise sintática LR:

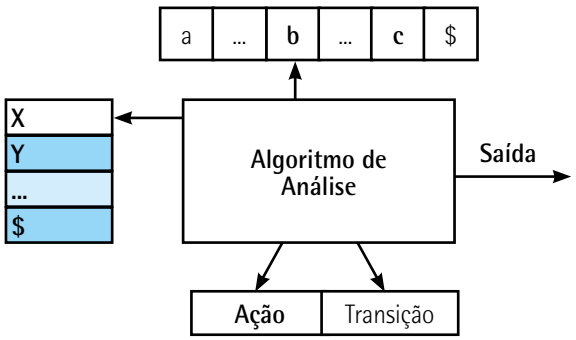

Figura 45 – Estrutura de um autômato de análise sintática LR

O algoritmo de análise emprega uma tabela de análise que apresenta linhas que se referem ao estado do processamento de uma cadeia de entrada e colunas de dois tipos: referentes a ações e referentes a transições.

As colunas referentes às ações são associadas aos símbolos terminais da gramática passíveis de serem encontrados na cadeia de entrada. Uma coluna adicional diz respeito ao símbolo indicador de fim de cadeia de entrada ($).

Assim, para cada par (estado, símbolo terminal da cadeia de entrada), corresponde uma possível ação, ora de empilhamento, ora de aplicação de regra.

A ação de empilhamento é indicada pelo símbolo ej, onde j é um número inteiro que indica qual estado deve ser empilhado juntamente com o símbolo da cadeia de entrada recém-lido. Pela ação de empilhamento, o símbolo terminal da entrada é inserido na pilha, sucedido pelo estado indicado.

A ação de aplicação de regra, indicada pelo símbolo rj, assinala qual regra j deverá ser aplicada, para que o conteúdo da pilha seja removido e se proceda a uma adequada inserção de um novo símbolo. Trata-se de uma ação de substituição de uma forma sentencial presente no lado direito da regra pelo não terminal no lado esquerdo da produção. Os estados justapostos aos símbolos terminais e não terminais na pilha são também removidos, permanecendo, porém, um estado à esquerda do símbolo não terminal recém-armazenado na pilha. Essa configuração da pilha, representada pelo par (estado, símbolo não terminal), determina a inserção de um novo estado na pilha, que é obtido a partir da consulta à tabela de análise, nas colunas referentes a transições. De fato, sempre que houver uma ação de aplicação de regra, o par (estado, símbolo não terminal) determina a transição para um estado que se realiza através da inserção do mesmo no topo pilha.

O algoritmo inicializa-se com a cadeia de entrada a ser analisada e a pilha com o estado q0 armazenado.

O autômato é finalizado quando atinge o par (estado, $), onde $ indica fim da cadeia de entrada, cuja ação especificada é aquela especial denominada ac, ou seja, de aceitação da cadeia de entrada.

Qualquer combinação de estado, símbolo que resulte em uma célula vazia na tabela de análise, corresponde a uma configuração de rejeição da cadeia de entrada.

A gramática com regras para expressões aritméticas é aqui novamente apresentada.

(1) E → E + T

(2) E → T

(3) T → T * F

(4) T → F

(5) F → (E)

(6) F → x

Apresenta-se a seguir a tabela de análise para a gramática de expressão:

Tabela 13 – Tabela para a gramática de expressão

S	Ação						Transição		
	x	+	*	(	)	$	E	T	F
0	e5			e4			1	2	3
1		e6				ac			
2		r2	e7		r2	r2			
3		r4	r4		r4	r4			
4	e5			e4			8	2	3
5		r6	r6		r6	r6			
6	e5			e4				9	3
7	e5			e4					10
8		e6			e11				
9		r1	r7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Fonte: Aho *et al.*, 2008.

A seguir, apresenta-se a sequência de movimentos do autômato para o processamento da cadeia de entrada x * (x + x).

Tabela 14

Pilha	Entrada	Ação
0	x*(x + x)$	O par (0, x) aponta na tabela para a ação de empilhamento e5: empilha símbolo x, recém-lido na cadeia de entrada e o estado 5.
0x5	*(x+x)$	O par (5, *) está associado à ação de redução r6: F → x. Assim, o símbolo x é substituído na pilha pelo símbolo F. Naturalmente o estado 5 também é removido. O par (0,F) desdobra-se na transição para o estado 3, conforme a tabela de análise. O estado 3 é armazenado na pilha.
0F3	*(x+x)$ *	O par (3, *) está associado à ação de redução r4: T → F. Assim, o símbolo F é substituído na pilha pelo símbolo T. Naturalmente o estado 3 também é removido. O par (0,T) desdobra-se na transição para o estado 2, conforme a tabela de análise. O estado 2 é armazenado na pilha.
0T2	*(x+x)$	O par (2, *) está associado à ação e7. O símbolo * e o estado 7 são armazenados na pilha.
0T2*7	(x+x)$	O par (7, ( ) está associado à ação e4. O símbolo (e o estado 4 são inseridos na pilha.
0T2*7(4	x+x)$	O par (4,x) está associado à ação e5. O símbolo x e o estado 5 são armazenados na pilha.
0T2*7(4x5	+x)$	O par (5, +) está associado à ação de redução r6: F → x. Assim, o símbolo x é substituído na pilha pelo símbolo F. Naturalmente o estado 5 também é removido. O par (4,F) desdobra-se na transição para o estado 3, conforme a tabela de análise. O estado 3 é armazenado na pilha.
0T2*7(4F3	+x)$	O par (3, +) resulta na ação de redução r4: T → F. Assim, o símbolo F e, consequentemente, o estado 3 são removidos da pilha e o símbolo T é inserido na pilha. Seguidamente, ocorre a inserção do estado 2, imposta pela ação de transição associada ao par 4,T.
0T2*7(4T2	+x)$	O par (2, +) resulta na ação de redução r2: E→ T. Assim, o símbolo T e, consequentemente, o estado 2 são removidos da pilha e o símbolo E é inserido na pilha. Seguidamente, ocorre a inserção do estado 8, imposta pela ação de transição associada ao par 4,E.
0T2*7(4E8	+x)$	O par (8, +) resulta na ação e6. O símbolo + e o estado 6 são inseridos na pilha.
0T2*7(4E8+6	x)$	O par (6, x) resulta na ação e5. O símbolo x e o estado 5 são inseridos na pilha.
0T2*7(4E8+6x5	)$	O par (5,)) resulta na ação de redução r6: F→ x. Assim, o símbolo x e, consequentemente, o estado 5 são removidos da pilha e o símbolo F é inserido na pilha. Seguidamente, ocorre a inserção do estado 3, imposta pela ação de transição associada ao par 6,F.
0T2*7(4E8+6F3	)$	O par (3,)) resulta na ação de redução r4: T→ F. Assim, o símbolo F e, consequentemente, o estado 3 são removidos da pilha e o símbolo T é inserido na pilha. Seguidamente, ocorre a inserção do estado 9, imposta pela ação de transição associada ao par 6,T.
0T2*7(4E8+6T9	)$	O par (9,)) resulta na ação de redução r1: E→E + T. Assim, a forma sentencial E8+6T9 é removida da pilha e substituída pelo símbolo E. Seguidamente, ocorre a inserção do estado 8, imposta pela ação de transição associada ao par 4,E.

0T2*7(4E8	)$	O par (8,)) resulta na ação e11. O símbolo) e o estado 11 são inseridos na pilha.
0T2*7(4E8)11	$	O par (11,$) resulta na ação de redução r5: F→ (E). Assim, a forma sentencial (4E8)11 é removida da pilha e o símbolo F é inserido na pilha. Seguidamente, ocorre a inserção do estado 10, imposta pela ação de transição associada ao par 7,F.
0T2*7F10	$	O par (10,$) resulta na ação de redução r3: T→ T*F. Assim, a forma sentencial T2*7F10 é removida da pilha e o símbolo T é inserido na pilha. Seguidamente, ocorre a inserção do estado 2, imposta pela ação de transição associada ao par 0,T.
0T2	$	O par (2,$) resulta na ação de redução r2: E→ T. Assim, a forma sentencial T2 é removida da pilha e o símbolo E é inserido na pilha. Seguidamente, ocorre a inserção do estado 1, imposta pela ação de transição associada ao par 0,E.
0E1	$	Finalmente o par (1, $) está associado ao estado de aceitação ac, ou seja, a cadeia de entrada x*(x+x) pode ser reduzida ao símbolo inicial E.

Lembrete

Se, por um lado, para toda Gramática Livre de Contexto existe um autômato de pilha não determinístico, nem sempre esse autômato corresponderá a um reconhecimento eficiente. Os métodos mais eficientes funcionam apenas para subclasses de gramáticas.

8 LINGUAGENS RECURSIVAS E LINGUAGENS RECURSIVAMENTE ENUMERÁVEIS

Neste item serão apresentadas as **linguagens sensíveis ao contexto** e as **linguagens recursivamente enumeráveis** que são geradas pelas **gramáticas sensíveis ao contexto** e pelas **gramáticas irrestritas**, respectivamente. O dispositivo reconhecedor dessas linguagens é a **máquina de Turing**.

As máquinas de Turing são similares aos autômatos finitos e de pilha e apresentam fita de entrada e unidade de controle finito. A figura a seguir ilustra a estrutura de uma máquina de Turing.

Note que a fita de entrada não é finita, e sim ilimitada à direita. Por outro lado, o seu início é marcado pelo símbolo especial •. Cumpre observar que há autores que apresentam a máquina de Turing como ilimitada à direita e à esquerda. Sobre a fita de entrada podem ser executadas as operações de leitura e escrita. Ainda, o cursor pode movimentar-se à direita e à esquerda. O símbolo gravado e o sentido do movimento são definidos pelo programa da unidade de controle.

O programa é uma função tal que, dependendo do estado corrente da máquina e do símbolo lido, são determinados: o próximo estado, o sentido do movimento do cursor e o símbolo a ser gravado na fita.

Como a fita é ilimitada à direita, inicialmente, a palavra a ser processada ocupa as células mais à esquerda, sendo que as demais são aqui denotadas por β, ou seja, células em branco.

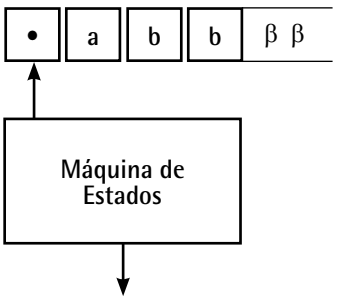

Figura 46 – Estrutura de uma máquina de Turing

O modelo da máquina de Turing foi proposto por Alan Turing em 1936, e ainda nesse ano Alonzo Church apresentou a **Hipótese de Church**, que pode ser enunciada como: "As máquinas de Turing são versões formais de algoritmos e nenhum procedimento computacional é considerado um algoritmo a não ser que possa ser apresentado na forma de uma máquina de Turing" (NETO, 2003).

Linguagens sensíveis a contexto: são definidas a partir das **gramáticas sensíveis a contexto.**

Seja G = (V, Σ, P, S), diz-se que G é sensível ao contexto se suas produções são da forma:

$\alpha \rightarrow \beta$

onde: $\alpha \in (V \cup \Sigma)^+$, $\beta \in (V \cup T)^*$ e $|\alpha| \leq |\beta|$, excetuando-se para a regra $S \rightarrow \varepsilon$.

Nesse caso, S não pode estar do lado direito de qualquer produção.

Um exemplo de uma linguagem sensível a contexto, definida sobre o alfabeto $\Sigma = \{a, b, c\}$, é:

$$L(w) = \{w \mid w = a^n b^n c^n, n > 0\}$$

Observe-se que uma memória auxiliar organizada em pilha não seria suficiente para verificar se os números de ocorrências dos símbolos a, b e c são iguais entre si. Para tanto, seriam necessárias ao menos uma máquina de estados com duas pilhas, equivalente à máquina de Turing.

Teorema: L é uma Linguagem Sensível ao Contexto se, e somente se, L é reconhecida por uma máquina de Turing com fita limitada.

Para a gramática irrestrita, as produções não apresentam restrições. Qualquer gramática G = (V, Σ, P, S) é uma gramática irrestrita.

Teorema: L é uma linguagem recursivamente enumerável se, e somente s, L é gerada por uma gramática irrestrita.

Segundo a Hipótese de Church, nenhum procedimento computacional é considerado um algoritmo a não ser que possa ser apresentado na forma de uma máquina de Turing. Assim sendo, pode-se afirmar que as classes das linguagens recursivamente enumeráveis são passíveis de serem processadas pelas máquinas de Turing.

Saiba mais

Para saber mais sobre a conceituação do autômato e do transdutor adaptativo, modalidades de dispositivos de reconhecimento e transdução sintática, leia:

NETO, J. J. *Contribuições à metodologia de construção de compiladores*. 1993. 306 f. Tese de Livre-Docência - Escola Politécnica da Universidade de São Paulo, São Paulo, 1993.

Nesta disciplina, das quatro classes de linguagens previstas na Hierarquia de Chomsky, procurou-se explanar de forma mais aprofundada as Linguagens Regulares e as Linguagens Livres de Contexto.

Para cada uma dessas duas linguagens, apresentaram-se os dispositivos reconhecedores, bem como os geradores das mesmas.

Deseja-se enfatizar que os resultados da Teoria da Computação para as Linguagens Regulares e para algumas subclasses das Linguagens Livres de Contexto garantem a existência de uma correspondência entre os dispositivos reconhecedores e os geradores das linguagens, com desempenho eficiente. Assim sendo, os algoritmos empregados em compiladores fazem uso dos dispositivos geradores e aceitadores próprios das Linguagens Regulares e Livres de Contexto.

Observe, no entanto, que o mesmo não se pode dizer com relação à componente dependente de contexto das linguagens de programação. Em vez de uma gramática dependente de contexto é empregada a Gramática de Atributos, que é na verdade uma extensão da Gramática Livre de Contexto (AHO *et al.*, 2008).

A aceitação das linguagens sensíveis ao contexto e das linguagens recursivamente enumeráveis pode ser realizada pela máquina de Turing. Na prática, no entanto, as implementações eficientes deste modelo conceitual dependem de formulações determinísticas equivalentes e, para tanto, isso é atualmente objeto de estudo da Teoria da Computação.

Em sua tese de livre-docência, Neto (1994) introduziu o conceito de autômatos adaptativos, dispositivos com poder equivalente ao da máquina de Turing. Esses dispositivos apresentam

a capacidade de representar, formalizar e, portanto, reconhecer linguagens dependentes de contexto e até mesmo conjuntos recursivamente enumeráveis. Em Ramos (2009), os dispositivos adaptativos são apresentados.

Resumo

Nesta unidade foram apresentadas as Linguagens Livres de Contexto, as Linguagens Sensíveis a Contexto e as Linguagens Recursivamente enumeráveis.

A Gramática Livre de Contexto foi definida por Chomsky em 1956 e trata-se de um dispositivo mais poderoso que a Gramática Regular. De fato, as produções permitem expressar como os elementos de uma forma sentencial se organizam em grupos hierárquicos complexos.

Os algoritmos empregados em compiladores fazem uso dos dispositivos geradores e aceitadores das Linguagens Livres de Contexto para o projeto e implementação do módulo funcional analisador sintático.

Exemplos de estruturas sintáticas em uma linguagem de programação são os diferentes comandos (*if*, *while*, classes, variáveis). Para a representação e o processamento dessas estruturas, devem-se empregar os estudos advindos das Linguagens Livres de Contexto.

As Linguagens Livres de Contexto são geradas pelas gramáticas livres de contexto.

Uma gramática livre de contexto é uma gramática $G = (V, \Sigma, P, S)$, onde as produções são da forma:

$A \rightarrow \alpha$, onde $A \in V$ e $\alpha \in (V \cup \Sigma)^*$.

São exemplos de Linguagens Livres de Contexto aquelas que apresentam duplo balanceamento, tais como:

$L(w) = \{w \mid w = a^n b^n , n \geq 0\}$

$L(w) = \{w \mid w = a^n b^n c^m d^m , n \geq 0, m > 0\}$

Uma árvore de derivação é uma representação gráfica de uma derivação que filtra a ordem na qual as produções são aplicadas para substituir símbolos não terminais.

Cumpre observar que toda Linguagem Regular é Livre de Contexto, mas nem toda Linguagem Livre de Contexto é regular. Portanto, em sentido estrito, ela não pode ser reconhecida por um autômato finito.

O dispositivo reconhecedor de uma Linguagem Livre de Contexto é o autômato de pilha, que pode ser representado por uma sêxtupla M:

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

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;

g é uma relação de transição $g : (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*) \times (Q \times \Gamma^*)$;

q_0 é o estado inicial;

$F \subseteq Q$ é o conjunto de estados finais.

Dois resultados importantes: se L é uma Linguagem Livre de Contexto, então: existe M, autômato de pilha, com controle de aceitação por estados finais com somente três estados, tal que M aceite L; existe M, autômato de pilha com controle de aceitação por pilha vazia, com somente um estado, tal que M aceite L.

As Linguagens Livres de Contexto geradas por Gramáticas Livres de Contexto e as linguagens aceitas por autômatos de pilha são equivalentes. Tal atributo proporciona dois métodos diferentes para se verificar se uma palavra pertence à linguagem gerada a partir das gramáticas.

Um analisador sintático de uma linguagem gerada por uma gramática verifica se é possível construir uma árvore de derivação para uma dada cadeia de entrada.

Segundo Aho *et al.* (2008), existem três estratégias gerais de análise sintática para o processamento de gramáticas: universal, descendente e ascendente.

Como exemplo de método universal, foi apresentado o algoritmo de Cocke-Youger-Kasami, que emprega a gramática na forma normal de

Chomsky. Também foi apresentado o método de análise ascendente, que emprega uma tabela de análise.

As máquinas de Turing são similares aos autômatos finitos e de pilha e apresentam fita de entrada e unidade de controle finito.

A fita de entrada de uma máquina de Turing é ilimitada à direita e permite que operações de leitura e escrita sejam efetuadas sobre a mesma. A unidade de controle apresenta um conjunto finito de estados, sendo que o cursor que aponta para os símbolos armazenados na fita de entrada pode se movimentar à direita e à esquerda.

Para o par, símbolo da fita de entrada lido, estado atual de processamento, uma função de transição determina o próximo estado a ser alcançado pela máquina, o símbolo a ser gravado na fita e o movimento do cursor a ser realizado, seguidamente.

Linguagens Sensíveis a Contexto: são definidas a partir das Gramáticas Sensíveis a Contexto.

Seja G = (V, Σ, P, S), diz-se que G é sensível ao contexto se suas produções são da forma:

$$\alpha \rightarrow \beta$$

onde: $\alpha \in (V \cup \Sigma)^+$, $\beta \in (V \cup T)^*$ e $|\alpha| \leq |\beta|$, excetuando-se para a regra $S \rightarrow \varepsilon$. Nesse caso, S não pode estar do lado direito de qualquer produção.

Um exemplo de uma Linguagem Sensível ao Contexto, definida sobre o alfabeto $\Sigma = \{a, b, c\}$, é:

$$L(w) = \{w \mid w = a^n b^n c^n, n > 0\}$$

L é uma Linguagem Sensível ao Contexto se, e somente se, L é reconhecida por uma máquina de Turing com fita limitada.

Para a gramática irrestrita, as produções não apresentam restrições. Qualquer gramática G = (V, Σ, P, S) é uma gramática irrestrita.

L é uma Linguagem Recursivamente Enumerável se, e somente se, L é gerada por uma gramática irrestrita.

REFERÊNCIAS

Textuais

AHO, A. V. *et al. Compiladores, princípios, técnicas e ferramentas*. 2. ed. São Paulo: Pearson Addison Wesley, 2007.

CHOMSKY, N. *Three Models for the Descriptions of Languages*. IRE Transactions on Information Theory, 1956. Disponível em: <http://www.chomsky.info/articles/195609--.pdf>. Acesso em: 12 fev. 2013.

GERSTING, J. L. *Fundamentos matemáticos para a Ciência da Computação.* 4. ed. Rio de Janeiro: LTC, 1999.

LEWIS, H. R.; PAPADIMITRIOU, C. H. *Elements of the Theory of Computation*. 2. ed. Upper Saddle River: Prentice Hall, 1998.

MENEZES, P. B. *Linguagens formais e autômatos*. Porto Alegre: Bookman, 2008.

NETO, J. J. *Contribuições à metodologia de construção de compiladores*. 1993. 306 f. Tese de Livre-Docência - Escola Politécnica, Universidade de São Paulo, São Paulo, 1993.

___. Adaptive Automata for Context – Sensitive Languages. *Sigplan notices*, v. 29, n. 9, p. 115-124, September, 1994. Disponível em: <http://lta.poli.usp.br/lta/publicacoes/artigos/1990-1999/neto-jj-9x-6. Acesso em: 1 abr. 2013.

___. *Tópicos sobre os fundamentos da Engenharia da Computação.* Disponível em: <http://www.pcs.usp.br/~pcs5701/2003/index.html>. Acesso em: 1 abr. 2013.

RAMOS, M. V. M.; NETO, J. J.; VEGA, I. S. *Linguagens formais*. Porto Alegre: Bookman, 2009.

RAPOSO, E. P. *Teoria da Gramática*. A faculdade da linguagem. Lisboa: Caminho, 1998.

ROSA, J. L. G. *Linguagens formais e autômatos*. Rio de Janeiro: LTC, 2010.

SEBESTA, R. W. *Conceitos de linguagens de programação*. 5. ed. Porto Alegre: Bookman, 2003.

Interativa

Informações:
www.sepi.unip.br ou 0800 010 9000