

CAPÍTULO 3 - LINGUAGENS LIVRES DE CONTEXTO

RECORDAÇÃO: uma gramática livre de contexto (GLC) é uma 4-upla $G = (N, \Sigma, P, S)$ onde:

N - conjunto de símbolos não-terminais

Σ - alfabeto terminal ($N \cap \Sigma = \emptyset$; $V = N \cup \Sigma$)

S - símbolo inicial ($S \in N$)

P - conjunto de produções da forma

$$A \rightarrow \alpha \quad ; \quad A \in N, \quad \alpha \in V^*$$

EXEMPLO: $G = (N, \Sigma, P, S)$

$$N = \{ S, \langle op \rangle, \langle var \rangle, \langle cle \rangle \}$$

$$\Sigma = \{ 1, 0, x, y, z, (,), +, * \}$$

$$P: \quad S \rightarrow \langle var \rangle \mid \langle cle \rangle \mid S \langle op \rangle S \mid (S)$$

$$\langle op \rangle \rightarrow + \mid *$$

$$\langle var \rangle \rightarrow x \mid y \mid z$$

$$\langle cle \rangle \rightarrow 0 \mid 1 \mid 0 \langle cle \rangle \mid 1 \langle cle \rangle$$

$$(x + 101) * y \in L(G).$$

ÁRVORES SINTÁTICAS E AMBIGUIDADE

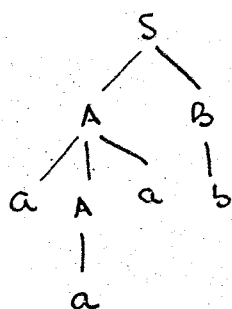
Seja GLC com produções :

$$S \rightarrow AB$$

$$A \rightarrow aAa \mid a$$

$$B \rightarrow b$$

$$(1) S \Rightarrow AB \Rightarrow aAaB \Rightarrow aaab \Rightarrow aaab$$

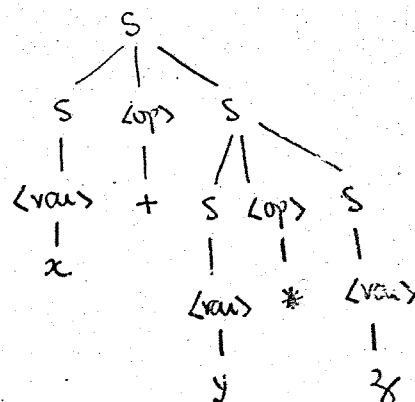
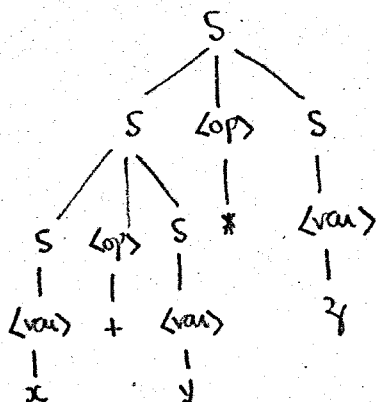


árvore sintática correspondente à derivação (1)

$$(2) S \Rightarrow AB \Rightarrow Ab \Rightarrow aAab \Rightarrow aaab$$

Essa derivação é diferente da derivação (1). No entanto, as árvores sintáticas associadas às duas derivações é a mesma.

Considere a gramática do exemplo anterior. Para a sentença $x+y*zy$ pode-se determinar as árvores sintáticas :

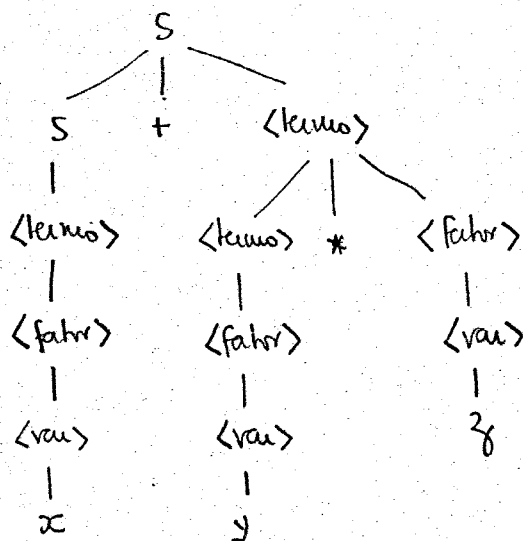


Neste caso existem duas árvores distintas para a mesma sentença (AMBIGÜIDADE)

Seja agora a gramática (equivalente à anterior):

$$\begin{aligned} S &\rightarrow S + \langle \text{termo} \rangle \mid \langle \text{termo} \rangle \\ \langle \text{termo} \rangle &\rightarrow \langle \text{termo} \rangle * \langle \text{fator} \rangle \mid \langle \text{fator} \rangle \\ \langle \text{fator} \rangle &\rightarrow \langle \text{var} \rangle \mid \langle \text{cte} \rangle \mid (S) \\ \langle \text{var} \rangle &\rightarrow x \mid y \mid z \\ \langle \text{cte} \rangle &\rightarrow 0 \mid 1 \mid 0 \langle \text{cte} \rangle \mid 1 \langle \text{cte} \rangle \end{aligned}$$

Para a sentença $x + y * z$ tem-se:



e esta árvore de derivação é única.

DERIVAÇÃO MAIS À ESQUERDA - substituir sempre o nó terminal mais à esquerda da forma sentencial.

DERIVAÇÃO MAIS À DIREITA - substituir sempre o nó terminal mais à direita da forma sentencial.

EXEMPLO: $E \rightarrow E+T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow (E) \mid a$

derivação mais à esquerda de $a+a$:

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

derivação mais à direita de $a+a$:

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

definição 3.1 - Uma gramática G é ambígua se existir uma sentença que possua 2 árvores sintáticas diferentes (ou 2 derivações mais à esquerda ou 2 derivações mais à direita)

definição 3.2 - Uma linguagem livre de contexto (LLC), L , é inerentemente ambígua se não existir gramática livre de contexto G , não ambígua, tal que $L = L(G)$.

EXEMPLO: $L = \{ a^i b^j c^k \mid i=j \text{ ou } j=k ; i, j, k \geq 1 \}$ é inerentemente ambígua.

INTUITIVAMENTE: L é inerentemente ambígua porque o processo de gerar sentenças de mesmo número de a 's e b 's é independente (e portanto diferente!) do processo

de gerar sentenças com mesmo número de b's e c's. Logo, é inevitável que para sentenças de mesmo número de a's, b's e c's, existam duas árvores sintáticas.

(para uma prova formal:

HARRISON, M.A. Introduction to formal language theory
pg. 233-239)

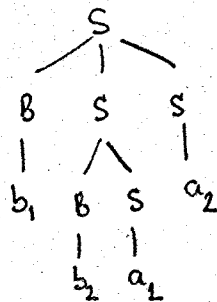
Exercício: Construir uma GLC que gere essa linguagem!

OUTRO EXEMPLO: "dangling else"

$S \rightarrow \text{if } B \text{ then } S \text{ else } S \mid \text{if } B \text{ then } S \mid a_1 \mid a_2$
 $B \rightarrow b_1 \mid b_2$

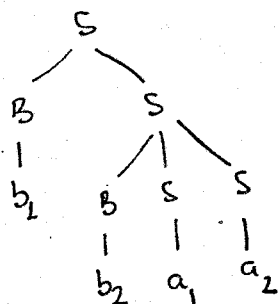
se for substituído por "if B then S"
dará ambiguidade.

Seja: if b_1 then if b_2 then a_1 else a_2



corresponde a:

if b_1 then
 if b_2 then a_1
else a_2



corresponde a:

if b_1 then
if b_2 then a_1 else a_2

Neste caso o programador e o compilador podem ter opiniões diferentes a respeito da estrutura. Em linguagens de programação a ambiguidade é resolvida por uma convenção "semântica": o else corresponde ao if mais próximo.

FORMA NORMAL DE CHOMSKY

definição 3.3. Uma GLC $G = (N, \Sigma, P, S)$ é \perp -livre se

- P não contém produções da forma $A \rightarrow \perp$, $A \in N$
- a única produção desse tipo for $S \rightarrow \perp$ e S não aparece no lado direito de nenhuma produção.

definição 3.4. Uma GLC $G = (N, \Sigma, P, S)$ está na forma normal de Chomsky (FNC) se G for \perp -livre e suas produções forem da forma

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

$$(A, B, C \in N ; a \in \Sigma)$$

EXEMPLO: Seja G_1 com produções:

$$S \rightarrow \perp \mid ab \mid aAb$$

$$A \rightarrow aAb \mid ab$$

G_1 é \perp -livre mas não está na FNC. Notar que

$$L(G_1) = \{ a^n b^n \mid n \geq 0 \}$$

Seja G_2 com produções:

$$S \rightarrow \perp \mid AB \mid AX$$

$$X \rightarrow YB$$

$$Y \rightarrow AX \mid AB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$S \Rightarrow AX \Rightarrow AYB \Rightarrow AAXB \Rightarrow AAYBB \Rightarrow AAABBB \stackrel{6}{\Rightarrow} aaabbb$$

G_2 está na FNC e $L(G_2) = L(G_1)$ (exercício!)

Por que o interesse pela forma normal de Chomsky?

Porque para uma gramática na FNC as árvores sintáticas são sempre árvores binárias (incompletas) e essa limitação na ramificação das árvores binárias simplifica muitas provas. Portanto, a FNC tem uma importância teórica.

Lema 3.1 ... Existe um algoritmo que transforma uma GLC $G = (N, \Sigma, P, S)$ qualquer, numa outra GLC $G' = (N', \Sigma, P', S')$ equivalente e \perp -livre.

Prova. Seja $\#N = n$ e $w_1 = \{A \mid A \rightarrow \perp \in P\}$

Para todo $k \geq 1$, seja

$$w_{k+1} = w_k \cup \{A \mid A \rightarrow \alpha_1 \dots \alpha_m; \alpha_i \in w_k, 1 \leq i \leq m\}$$

Tem-se então:

$$(1) \quad w_i \subseteq w_{i+1} \quad \text{para todo } i \geq 1$$

$$(2) \quad \text{se } w_i = w_{i+1} \text{ então } w_i = w_{i+m}, \quad m \geq 1$$

$$(3) \quad w_{n+1} = w_n, \text{ ou seja, a construção dos conjuntos } w_i \text{ pára, na pior das hipóteses, quando } i = n.$$

$$(4) \quad w_n = \{A \in N \mid A \xRightarrow{*} \perp\}$$

$$(5) \quad \perp \in L(G) \iff S \in w_n$$

Então, para construir a gramática $G' = (N \cup \{S'\}, \Sigma, P', S')$ seja P' com produções:

$$a) \quad S' \rightarrow S$$

$$b) \quad S' \rightarrow \perp \in P' \iff S \in w_n$$

$$c) \quad A \rightarrow A_1 \dots A_k, \quad k \geq 1, \quad A_i \in N \cup \Sigma \text{ tal que}$$

existem $\alpha_1, \dots, \alpha_{k+1} \in \omega_n^*$ tal que $A \rightarrow \alpha_1 A_1 \dots \alpha_k A_k \alpha_{k+1} \in P$

Claramente, S' não aparece no lado direito de qualquer produção em P' .
 Além disso,

$$\perp \in L(G') \Leftrightarrow S' \rightarrow \perp \in P' \Leftrightarrow S \in \omega_n \Leftrightarrow S \xRightarrow{*} \perp \Leftrightarrow \perp \in L(G)$$

Como em G' , não existem produções levando a \perp (exceto, possivelmente $S' \rightarrow \perp$), então G' é \perp -livre. De-se mostrar que $L(G') = L(G)$.

1) se $A \rightarrow A_1 \dots A_k \in P'$ então $A \rightarrow \alpha_1 A_1 \alpha_2 \dots \alpha_k A_k \alpha_{k+1} \in P$

e $\alpha_i \xRightarrow[G]{*} \perp$, $i = 1, \dots, k+1$. Logo:

$$A \xRightarrow[G]{*} \alpha_1 A_1 \dots \alpha_k A_k \alpha_{k+1} \xRightarrow[G]{*} A_1 \dots A_k$$

Portanto, toda produção $A \rightarrow A_1 \dots A_k \in P'$ pode ser simulada em G por uma derivação. Logo:

$$L(G') \subseteq L(G).$$

2) Será mostrado por indução no comprimento da derivação que se:

$$A \xRightarrow[G]{h} \beta, \beta \in \Sigma^+ \text{ então } A \xRightarrow[G']{*} \beta$$

base: $h=1$. Neste caso, $A \rightarrow \beta \in P$. Logo, como $\beta \neq \perp$,

$$A \rightarrow \beta \in P' \quad \left(\text{porque é sempre possível escolher } \alpha_1 = \dots = \alpha_{k+1} = \perp \in \omega_n^* \right).$$

hipótese indutiva : $A \xRightarrow[h]{h} \beta$ então $A \xRightarrow[q']{*} \beta$, $h \geq 1$.

passo da indução :

Seja $A \Rightarrow_{\bar{q}} A_1 \dots A_k \xRightarrow[h]{h} \beta$, $\beta \in \Sigma^+$.

Então existem $\beta_i \in \Sigma^*$ tais que $\beta = \beta_1 \dots \beta_k$ e $A_i \xRightarrow[h']{h'} \beta_i$.

com $h' \leq h$. Pela hipótese indutiva , $A_i \xRightarrow[q']{*} \beta_i$, $i=1, \dots, k$.

Se $\beta_i = \perp$ então $A_i \in \omega_n$ e $A \rightarrow A_{j_1} \dots A_{j_p} \in \mathcal{P}'$ onde

$A_{j_1} \dots A_{j_p}$ é uma subsequência de A_1, \dots, A_k obtida restando-se A_i .
Logo, em qualquer caso :

$$A \Rightarrow_{\bar{q}'} A_{j_1} \dots A_{j_p} \xRightarrow[q']{*} \beta_{j_1} \dots \beta_{j_p} = \beta \quad . \quad \text{Logo } L(\bar{q}) \subseteq L(\bar{q}').$$

Portanto, de 1 e 2, vem que $L(\bar{q}') = L(\bar{q})$.

EXEMPLO: Seja GLC com produções :

$$\begin{aligned} S &\rightarrow aAb \\ A &\rightarrow P \mid aAb \mid AA \\ P &\rightarrow x \mid E \\ E &\rightarrow \perp \end{aligned}$$

Então :

$$\omega_1 = \{ E \}$$

$$\omega_2 = \{ E, P \}$$

$$\omega_3 = \{ E, P, A \} = \omega_4$$

Para construir G' (Λ -livre e equivalente a G) deve-se lembrar que:

- toda produção de G deve estar em G' exceto produções do tipo $A \rightarrow \Lambda$, pois é sempre possível escolher $\alpha_1, \dots, \alpha_{k+1}$ todos iguais a Λ .
- uma produção de P' pode ser obtida de uma produção de P retirando do lado direito, um não terminal que pertença a w_n , com o cuidado de não tornar o lado direito vazio.

Logo: $G' = (\{S, A, P, E, S'\}, \{a, b, x\}, P', S')$, com P' dado por:

por:

$$S' \rightarrow S$$

$$S \rightarrow aAb \mid ab$$

$$A \rightarrow P \mid aAb \mid AA \mid ab$$

$$P \rightarrow x \mid E.$$

Lema 3.2. Existe um algoritmo que transforma qualquer gramática livre de contexto $G = (N, \Sigma, P, S)$ em outra $G' = (N, \Sigma, P', S)$ equivalente, Λ -livre e sem produções do tipo $A \rightarrow B$, $A, B \in N$ (produção inótil).

Prova (INFORMAL). Pelo lema 3.1, pode-se admitir G Λ -livre.

ALGORITMO:

(1) construir para cada $A \in N$, o conjunto

$$N_A = \{B \in N \mid A \xRightarrow{*} B\} \text{ da seguinte maneira:}$$

$$a) \quad N_0 = \{A\}; \quad i=1.$$

$$b) N_i = \{ C / B \rightarrow C \in P, B \in N_{i-1} \} \cup N_{i-1}$$

c) se $N_i \neq N_{i-1}$ então fazer $i = i+1$ e voltar para (b).
Caso contrário, $N_A = N_i$.

(2) construir P' da seguinte forma:

se $B \rightarrow \alpha \in P$ e não é da forma $B \rightarrow A$, colocar $A \rightarrow \alpha$ em P' , para todo A tal que $B \in N_A$.

EXEMPLO:

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

$$N_E = \{ E, T, F \}$$

$$N_T = \{ T, F \}$$

$$N_F = \{ F \}$$

A nova gramática será:

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

Teorema 3.1. Seja L uma LLC. Então, existe G na FNC tal que $L = L(G)$.

Prova. Pelos lemas 3.1 e 3.2, $L = L(G)$ para alguma GLC,

$G = (N, \Sigma, P, S)$, Σ -livre e sem produções da forma $A \rightarrow B$.

Construir G' na FNC da seguinte maneira:

$G' = (N', \Sigma, P', S)$ tal que P' é dado por:

- (1) $\text{se } A \rightarrow a \in P$ então $A \rightarrow a \in P'$ ($a \in \Sigma$)
 (2) $\text{se } A \rightarrow BC \in P$ então $A \rightarrow BC \in P'$ ($B, C \in N$)
 (3) $\text{se } S \rightarrow \perp \in P$ então $S \rightarrow \perp \in P'$

- (4) $\text{se } A \rightarrow X_1 \dots X_k \in P$ ($k > 2$), adicionar a P' as produções e os não terminais seguintes:

$$\begin{aligned} A &\rightarrow X'_1 \langle X_2 \dots X_k \rangle \\ \langle X_2 \dots X_k \rangle &\rightarrow X'_2 \langle X_3 \dots X_k \rangle \\ &\vdots \\ \langle X_{k-1} X_k \rangle &\rightarrow X'_{k-1} X'_k \end{aligned}$$

onde:

$$X'_i = \begin{cases} X_i & \text{se } X_i \in N \\ \text{um novo não terminal} & \text{se } X_i \in \Sigma \end{cases}$$

- (5) para todo não terminal X'_i criado em (4), acrescentar a P' a produção $X'_i \rightarrow X_i$

EXERCÍCIO: Mostre que $L(G') = L(G)$!

EXEMPLO: Passar para a FNC:

$$\begin{aligned} S &\rightarrow aAB \mid BA \\ A &\rightarrow BBB \mid a \\ B &\rightarrow AS \mid b \end{aligned}$$

Neste caso tem-se:

$$S \rightarrow \langle a \rangle \langle AB \rangle \mid BA$$

$$A \rightarrow B \langle BB \rangle \mid a$$

$$B \rightarrow AS \mid b$$

$$\langle a \rangle \rightarrow a$$

$$\langle AB \rangle \rightarrow AB$$

$$\langle BB \rangle \rightarrow BB$$

O teorema a seguir é uma ferramenta importante para mostrar que certas linguagens não são livres de contexto.

Teorema 3.2 ("pumping lemma") Para toda linguagem livre de contexto L , podem ser determinados inteiros k e l tais que, se $z \in L$, $|z| > k$ então:

- 1) $z = uvwxy$
- 2) $|vwx| \leq l$
- 3) $vx \neq \epsilon$
- 4) para todo $i \geq 0$, $uv^iwx^iy \in L$.

Prova. Seja $L = L(G)$ para alguma gramática G na FNC. Logo as árvores de derivação em G são árvores binárias.

Fato 1: Se o caminho mais longo na árvore de derivação de z possuir ordem m então $|z| \leq 2^{m-1}$.

Porque: $|z|$ = número de "folhas" da árvore de derivação de z .
Numa árvore binária de ordem m existem, no máximo, 2^m folhas (o número de folhas não é 2^m , caso a árvore binária for completa). Entretanto, as árvores de derivação de ordem m de uma gramática na FNC possuem no máximo, 2^{m-1} folhas, porque da penúltima ordem para a última, em qualquer caminho, não há ramificação devido às produções $A \rightarrow a$.

Fato 2. Se $|z| > 2^m$ então algum caminho na árvore de derivação de z possui, no mínimo, ordem igual a $m+2$.

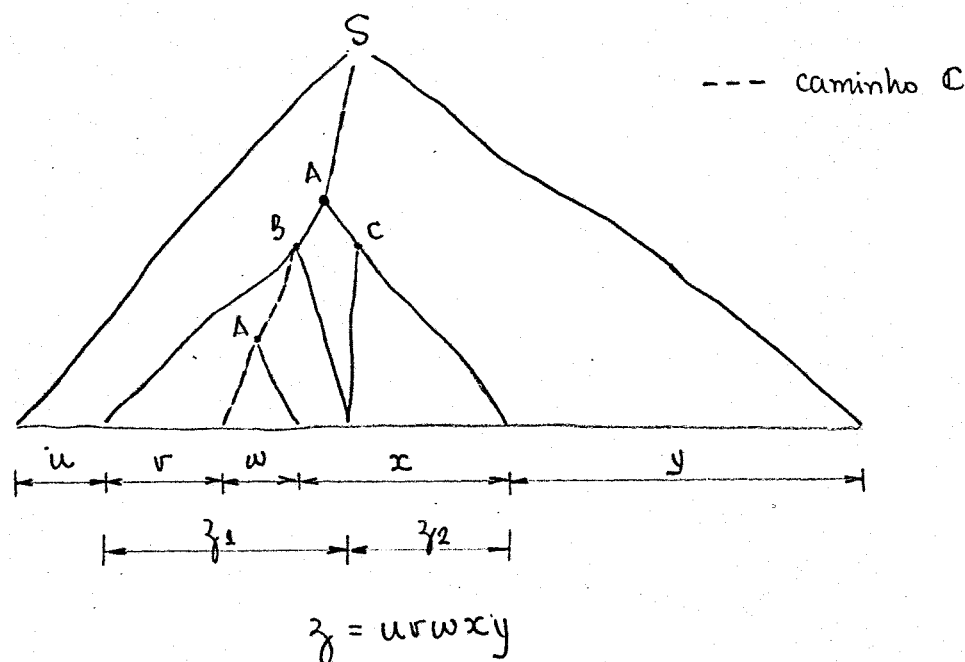
Porque. Se não existir caminho de ordem, no mínimo, $m+2$, o caminho mais longo na árvore de derivação de z terá, no máximo, ordem $m+1$ e pelo Fato 1, $|z| \leq 2^m$.

Sejam então: $k = 2^{n-1}$; $l = 2^n$ com $n = \#N$ (número de não terminais de G)

Seja $z \in L$, $|z| > k$. Para a árvore de derivação de z tem-se:

- pelo Fato 2, algum caminho tem ordem, pelo menos, $n+1$. Seja C o caminho de maior comprimento da árvore de derivação de z . Logo, C tem pelo menos $n+2$ nós.

- sejam os últimos $n+2$ nós de C . Logo existem $n+1$ não terminais neste sub-caminho. Como existem apenas n não terminais, deve existir pelo menos, um não terminal repetido. Seja A um não terminal repetido neste subcaminho, tal que:



Sobre o subcaminho de C a partir do primeiro A , pode-se dizer:

- tem, no máximo, $n+2$ nós
- a ordem é no máximo $n+1$
- a partir de A , não existe caminho mais longo que esse.

Logo:

- 1) $S \stackrel{*}{\Rightarrow} u A y \stackrel{*}{\Rightarrow} u v A x y \stackrel{*}{\Rightarrow} u r w x y = z$
- 2) $A \stackrel{*}{\Rightarrow} v A x \stackrel{*}{\Rightarrow} v w x$. Como a ordem desse caminho é, no máximo, $n+1$, pelo Fato 1, $|v w x| \leq 2^n = l$.
- 3) $A \Rightarrow B C \stackrel{*}{\Rightarrow} v w x = z_1 z_2$ onde $B \stackrel{*}{\Rightarrow} z_1$ e $C \stackrel{*}{\Rightarrow} z_2$

Então, $z_1 \neq \perp$ e $z_2 \neq \perp$ pois q está na FNC e portanto q é \perp -livre.

Logo: $x \neq \Lambda$ e portanto $vx \neq \Lambda$ (deve-se observar que foi suposto que a repetição de A se dá no caminho a partir de B. Se a repetição fosse no caminho a partir de C, então $v \neq \Lambda$. Em qualquer caso, $vx \neq \Lambda$).

$$4) A \xRightarrow{*} vAx \xRightarrow{*} vvAx \xRightarrow{*} v^iAx^i \quad (i \geq 0)$$

Por outro lado,

$$A \xRightarrow{*} w$$

Logo:

$$S \xRightarrow{*} uAy \xRightarrow{*} uv^iAx^iy \xRightarrow{*} uv^iw x^iy$$

Portanto, $uv^iw x^iy \in L$.

EXEMPLO: $L = \{ a^n b^n c^n \mid n \geq 0 \}$ não é LLC

Considere que L é LLC. Então para n suficientemente grande

$$|a^n b^n c^n| > k \quad (k \text{ do "pumping lemma"})$$

Logo, pode-se escrever:

$$a^n b^n c^n = uvwxy, \quad vx \neq \Lambda \text{ e}$$

$$uv^iw x^iy \in L, \quad i \geq 0.$$

Admitindo (sem perda de generalidade) que $v \neq \Lambda$, tem-se:

(1) $v = a^k$. Neste caso $uv^2wx^2y \notin L$, porque para equilibrar o número de a's, b's e c's, a subcadeia x deve ter k b's e k c's e portanto x^2 vai "embaralhar" os b's e c's.

O mesmo argumento vale para $v = b^k$ ou $v = c^k$.

(2) para qualquer outra forma, r terá pelo menos dois símbolos diferentes, por exemplo

$$r = a^p b^q.$$

Neste caso, novamente

$uv^2wx^2y \notin L$ porque existirá símbolos misturados. Por exemplo, se $r = a^p b^q$ então

uv^2wx^2y será da forma $\underbrace{a \dots a}_u \underbrace{a \dots a b \dots b}_v \underbrace{a \dots a b \dots b}_v \dots$

FORMA NORMAL DE GREIBACH

definição 3.5. um não terminal A numa GLC $G = (N, \Sigma, P, S)$ é recursivo à esquerda se $A \xrightarrow{+} A\beta$, $\beta \in (N \cup \Sigma)^*$. Uma gramática com pelo menos um não terminal recursivo à esquerda é uma GRAMÁTICA RECURSIVA À ESQUERDA.

Alguns algoritmos de análise sintática não trabalham com gramáticas recursivas à esquerda. Será mostrado a seguir que toda LLC tem pelo menos uma gramática não recursiva à esquerda que a gera.

Lema 3.3. Seja $G = (N, \Sigma, P, S)$ uma GLC na qual

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n$$

são todas as A -produções de P e nenhum β_i ($i=1, \dots, n$) começa com A . Seja $G' = (N \cup \{A'\}, \Sigma, P', S)$ onde

P' é obtido substituindo-se as A -produções de P por:

$$A \rightarrow \beta_1 | \beta_2 | \dots | \beta_n | \beta_1 A' | \dots | \beta_n A'$$

$$A' \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_m | \alpha_1 A' | \dots | \alpha_m A'$$

$$\text{Então, } L(G') = L(G).$$

Prova. Exercício!

Exemplo:

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

Eliminando-se a recursão à esquerda tem-se:

$$\begin{aligned} E &\rightarrow T \mid TE' \\ E' &\rightarrow +T \mid +TE' \\ T &\rightarrow F \mid FT' \\ T' &\rightarrow *F \mid *FT' \\ F &\rightarrow (E) \mid a \end{aligned}$$

ALGORITMO PARA ELIMINAÇÃO DE RECURSÃO À ESQUERDA

Seja $G = (N, \Sigma, P, S)$ uma GLC tal que $N = \{A_1, \dots, A_n\}$.

O algoritmo irá transformar G de tal maneira que suas produções tenham a forma $A_i \rightarrow \alpha$, onde α começa com um terminal ou com A_j , $j > i$, ou então $A'_i \rightarrow \beta$, onde A'_i é um novo não terminal e β começa com um símbolo de $(N \cup \Sigma)$.

(1) $i = 1$

(2) Sejam as A_i -produções na forma :

$$A_i \rightarrow A_i \alpha_1 | \dots | A_i \alpha_m | \beta_1 | \dots | \beta_\ell$$

onde nenhum β_k ($k = 1, \dots, \ell$) começa com A_j , $j \leq i$ (é sempre possível deixar as A_i -produções nessa forma)

Trocar as A_i -produções por :

$$A_i \rightarrow \beta_1 | \dots | \beta_\ell | \beta_1 A'_i | \dots | \beta_\ell A'_i$$

$$A'_i \rightarrow \alpha_1 | \dots | \alpha_m | \alpha_1 A'_i | \dots | \alpha_m A'_i$$

onde A'_i é um novo não terminal (após essa troca, todas as A_i -produções começam com um terminal ou com A_j , $j > i$)

Se $i = n$, então para. Caso contrário, $i = i + 1$; $j = 1$

(3) Sejam $A_j \rightarrow \gamma_1 | \dots | \gamma_p$ todas as A_j -produções.
Trocar toda produção $A_i \rightarrow A_j \delta$ pelas produções :

$$A_i \rightarrow \gamma_1 \delta | \dots | \gamma_p \delta$$

(após essa troca, todas as A_j -produções começam com um terminal ou com A_k , $k > j$)

Se $j = i - 1$, então voltar para (2). Caso contrário, $j = j + 1$ e voltar para (3).

EXEMPLO : eliminar a recursão à esquerda de

$$A_1 \rightarrow A_2 A_3 \mid a$$

$$A_2 \rightarrow A_3 A_1 \mid A_1 b$$

$$A_3 \rightarrow A_1 A_2 \mid A_3 A_3 \mid a$$

(passo 2 ; $i=1$) : as A_1 -produções já estão na forma desejada

(passo 3 ; $i=2, j=1$) : $A_2 \rightarrow A_3 A_1 \mid A_2 A_3 b \mid ab$

(passo 2 ; $i=2$) : $A_2 \rightarrow A_3 A_1 \mid ab \mid A_3 A_1 A'_2 \mid ab A'_2$

$$A'_2 \rightarrow A_3 b \mid A_3 b A'_2$$

(passo 3 ; $i=3, j=1$) : $A_3 \rightarrow A_3 A_3 \mid a \mid A_2 A_3 A_2 \mid a A_2$

($i=3, j=2$) :

$$A_3 \rightarrow A_3 A_3 \mid a \mid a A_2 \mid A_3 A_1 A_3 A_2 \mid ab A_3 A_2 \mid A_3 A_1 A'_2 A_3 A_2 \mid ab A'_2 A_3 A_2$$

(passo 2 ; $i=3$) :

$$A_3 \rightarrow a \mid a A_2 \mid ab A_3 A_2 \mid ab A'_2 A_3 A_2 \mid a A'_3 \mid a A_2 A'_3 \mid ab A_3 A_2 A'_3 \mid ab A'_2 A_3 A_2 A'_3$$

$$A'_3 \rightarrow A_3 \mid A_1 A_3 A_2 \mid A_1 A'_2 A_3 A_2 \mid A_3 A'_3 \mid A_1 A_3 A_2 A'_3 \mid A_1 A'_2 A_3 A_2 A'_3$$

ou seja :

$$A_1 \rightarrow A_2 A_3 \mid a$$

$$A_2 \rightarrow A_3 A_1 \mid ab \mid A_3 A_1 A'_2 \mid ab A'_2$$

$$A'_2 \rightarrow A_3 b \mid A_3 b A'_2$$

$$A_3 \rightarrow a \mid a A_2 \mid ab A_3 A_2 \mid ab A'_2 A_3 A_2 \mid a A'_3 \mid a A_2 A'_3 \mid$$

$$ab A_3 A_2 A'_3 \mid ab A'_2 A_3 A_2 A'_3$$

$$A'_3 \rightarrow A_3 \mid A_1 A_3 A_2 \mid A_1 A'_2 A_3 A_2 \mid A_3 A'_3 \mid A_1 A_3 A_2 A'_3 \mid A_1 A'_2 A_3 A_2 A'_3$$

definição 3.6 : Uma GLC $G = (N, \Sigma, P, S)$ está na forma normal de Greibach, se G é Λ -livre e toda produção de P é da forma $A \rightarrow a\alpha$; $a \in \Sigma$, $\alpha \in N^*$.

ALGORITMO PARA CONVERSÃO PARA A FORMA NORMAL DE GREIBACH

Seja $G = (N, \Sigma, P, S)$ uma GLC não recursiva à esquerda.
 Seja $<$ uma relação de ordenação parcial tal que toda A -produção começa com um terminal ou com um não terminal B tal que $A < B$.
 Seja $N = \{A_1, \dots, A_n\}$ tal que $A_1 < A_2 < \dots < A_n$.

(1) para $i = n-1$ até 1 fazer:

trocar toda produção da forma $A_i \rightarrow A_j \alpha$, com $A_i < A_j$ por $A_i \rightarrow \beta_1 \alpha \mid \dots \mid \beta_m \alpha$, onde $A_j \rightarrow \beta_1 \mid \dots \mid \beta_m$ são todas as A_j -produções.

(2) após as trocas do passo (1), todas as produções (exceto possivelmente $S \rightarrow \perp$) começam com um terminal. Para cada produção $A \rightarrow a X_1 \dots X_k$, trocamos os X_j ($j=1, \dots, k$) que são terminais por X'_j ($j=1, \dots, k$) (novos não terminais) e adicionamos a produção $X'_j \rightarrow X_j$.

EXEMPLO: Seja a GLC não recursiva à esquerda:

$$\begin{aligned} E &\rightarrow T \mid TE' \\ E' &\rightarrow +T \mid +TE' \\ T &\rightarrow F \mid FT' \\ T' &\rightarrow *F \mid *FT' \\ F &\rightarrow (E) \mid a \end{aligned}$$

Seja a ordenação sobre os não-terminais: $E' < E < T' < T < F$

Como F é o "maior" na ordenação, todas as F -produções começam com um terminal. O próximo na ordenação é T . As T -produções são:

$$T \rightarrow F \mid FT'$$

Substituindo-se F nessas produções tem-se:

$$T \rightarrow (E) \mid a \mid (E)T' \mid aT'$$

Continuando:

$$T' \rightarrow *F \mid *FT'$$

$$E \rightarrow (E) \mid a \mid (E)T' \mid aT' \mid (E)E' \mid aE' \mid (E)T'E' \mid aT'E'$$

$$E' \rightarrow +T \mid +TE'$$

Após essas transformações, todas as produções começam com um terminal. Basta agora transformar os outros símbolos do lado direito das produções em não-terminais, introduzindo a produção $P \rightarrow)$. A gramática na forma normal de Greibach resultante será:

$$E \rightarrow (EP \mid a \mid (EPT' \mid aT' \mid (EPE' \mid aE' \mid (EPT'E' \mid aT'E'$$

$$E' \rightarrow +T \mid +TE'$$

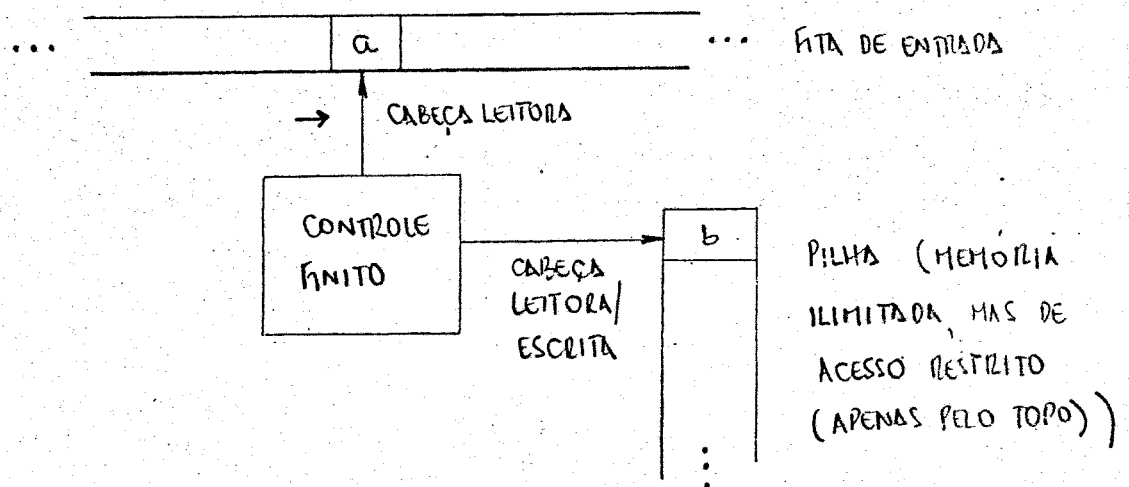
$$T \rightarrow (EP \mid a \mid (EPT' \mid aT'$$

$$T' \rightarrow *F \mid *FT'$$

$$F \rightarrow (EP \mid a$$

$$P \rightarrow)$$

AUTÔMATOS DE PILHA ("pushdown automata")



Intuitivamente :

Dependendo do :

- estado do controle finito
- símbolo que está sendo lido na fita de entrada
- símbolo no topo da pilha

o autômato de pilha :

- muda de estado
- escreve um número finito de símbolos na pilha (escrever \perp corresponde a apagar o símbolo no topo)
- move sua cabeça leitora uma posição para a direita

O autômato de pilha pode também efetuar \perp -movimentos, que corresponde a mudar de estado e mudar o conteúdo da pilha, sem ler o símbolo na fita de entrada (isto é, sem mover sua cabeça leitora para a direita)

definição 3.7 . Um autômato de pilha é uma 7-upla
 $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ onde :

Q - conjunto finito não vazio de estados

Σ - alfabeto de entrada

Γ - alfabeto da pilha

$q_0 \in Q$ - estado inicial

$z_0 \in \Gamma$ - símbolo inicial da pilha

$F \subseteq Q$ - conjunto de estados finais

$$\delta : Q \times (\Sigma \cup \{\perp\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*) \quad (\text{não determinístico})$$

definição 3.8. Uma configuração de um autômato de pilha $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ é um elemento de $Q \times \Sigma^* \times \Gamma^*$.

definição 3.9. (transição do autômato de pilha)

$$(q, ax, zw) \vdash (q', x, yw) \Leftrightarrow (q', y) \in \delta(q, a, z)$$

$$\text{com: } q, q' \in Q; a \in \Sigma \cup \{\perp\}; x \in \Sigma^*; w \in \Gamma^*; \\ z \in \Gamma; y \in \Gamma^*$$

Deve-se notar que pela definição 3.9, o autômato de pilha não faz transições com a pilha vazia.

definição 3.10. A linguagem aceita por $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, por estado final é:

$$L(A) = \{w \in \Sigma^* / (q_0, w, z_0) \vdash^* (q, \perp, x); \\ q \in F; x \in \Gamma^*\}$$

definição 3.11. A linguagem aceita por $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, por pilha vazia é:

$$N(A) = \{w \in \Sigma^* / (q_0, w, z_0) \vdash^* (q, \perp, \perp); q \in Q\}$$

EXEMPLO: Construir autômato de pilha que aceita

$$L = \{0^n 1^n / n \geq 0\}$$

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \{z, \perp\}, \delta, q_0, z, \{q_0\})$$

onde δ é dada por:

$$\delta(q_0, 0, z) = \{(q_1, 0z)\} \quad \left| \quad \begin{array}{l} \text{(empilha } 0's) \end{array} \right.$$

$$\delta(q_1, 0, 0) = \{(q_1, 00)\}$$

$$\delta(q_1, 1, 0) = \{(q_2, \perp)\} \quad \left| \quad \begin{array}{l} \text{(para cada 1 encontra-} \\ \text{do, desempilha um} \\ \text{0)} \end{array} \right.$$

$$\delta(q_2, 1, 0) = \{(q_2, \perp)\}$$

$$\delta(q_2, \perp, z) = \{(q_0, z)\}$$

EXERCÍCIO: Mostre que $L = L(A)$!

definição 3.12: Um autômato de pilha $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ é determinístico se para todo

$(q, a, z) \in Q \times (\Sigma \cup \{\perp\}) \times \Gamma$
tem-se:

$$1) \quad |\delta(q, a, z)| \leq 1$$

$$2) \quad \text{se } \delta(q, \perp, z) \neq \emptyset \text{ então } \delta(q, a, z) = \emptyset, \forall a \in \Sigma.$$

Observar que o autômato A do exemplo acima é determinístico.
Considere, no entanto a linguagem

$$L = \{ww^R \mid w \in \{0, 1\}^*; w^R \text{ é o reverso de } w\}$$

Inibitivamente, um autômato com pilha para reconhecer L deverá

ser não determinístico, porque durante o processamento de uma cadeia ww^R não há maneira de saber quando termina a cadeia w e começa a cadeia w^R . Assim, para cada símbolo lido, o autômato deve prever duas possíveis situações:

- a cadeia w ainda não terminou
- a cadeia w terminou e os próximos símbolos são de w^R .

Para a linguagem $L' = \{ ww^R \mid w \in \{0,1\}^* \}$, pode-se construir um autômato de pilha determinístico que a reconhece.

EXERCÍCIO: Construir um autômato de pilha que reconheça L !

Definição 3.13. Seja Σ um alfabeto e A um autômato de pilha.

$$L_\Sigma = \{ L \subseteq \Sigma^* \mid L = L(A) \} \quad (\text{conjunto das linguagens aceitas por estado final})$$

$$N_\Sigma = \{ L \subseteq \Sigma^* \mid L = N(A) \} \quad (\text{conjunto das linguagens aceitas por pilha vazia})$$

Proposição 3.1: $L_\Sigma = N_\Sigma$.

Lema 3.4. Se $L = L(A)$, para algum autômato de pilha A , então existe autômato de pilha B tal que $L = N(B)$.

Prova. Seja $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$.

Construir $B = (Q \cup \{d, q'_0\}, \Sigma, \Gamma \cup \{y_0\}, \delta_B, q'_0, y_0, \phi)$
com δ_B dada por:

$$(1) \quad \delta_B(q'_0, \perp, y_0) = \{(q_0, z_0 y_0)\}$$

(B se prepara para simular A, colocando-se na mesma situação inicial de A, a menos de y_0 na pilha)

$$(2) \quad (q', \alpha) \in \delta(q, a, z) \Rightarrow (q', \alpha) \in \delta_B(q, a, z)$$

$$q \in Q; a \in \Sigma \cup \{\perp\}; z \in \Gamma$$

(ou seja, B simula A)

$$(3) \quad (d, \perp) \in \delta_B(q, \perp, z) \quad ; \quad q \in F \quad ; \quad z \in \Gamma$$

$$(4) \quad \delta_B(d, \perp, z) = \{(d, \perp)\}$$

(ou seja, sempre que o autômato A entrar num estado final, o autômato B divide-se em duas cópias: uma delas continua a simular A (transições devidas a (2)) e a outra esvazia a pilha (transições devidas a (3) e (4)).

$$\text{Então:} \quad x \in L(A) \Leftrightarrow (q_0, x, z_0) \stackrel{*}{\vdash}_A (q, \perp, w); \quad q \in F, \quad w \in \Gamma^*$$

$$\Leftrightarrow (q'_0, x, y_0) \vdash_B (q_0, x, z_0 y_0) \quad (\text{devido a (1)})$$

$$\stackrel{*}{\vdash}_B (q, \perp, w y_0) \quad (\text{devido a (2)})$$

$$\stackrel{*}{\vdash}_B (d, \perp, \perp) \quad (\text{devido a (3) e (4)})$$

$$\Leftrightarrow x \in N(B).$$

Logo, $L(A) = N(B)$ e portanto $L_{\Sigma} \subseteq N_{\Sigma}$.

Lema 3.5. Para todo autômato de pilha A , existe um autômato de pilha B tal que $L(B) = N(A)$.

Prova. Seja $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$.

Construir $B = (Q \cup \{f, q'_0\}, \Sigma, \Gamma \cup \{y_0\}, \delta_B, q'_0, y_0, \{f\})$
com δ_B dada por:

- 1) $\delta_B(q'_0, \perp, y_0) = \{(q_0, z_0 y_0)\}$
- 2) $\delta_B(q, a, \alpha) = \delta(q, a, \alpha)$; $q \in Q, a \in \Sigma \cup \{\perp\}, \alpha \in \Gamma$
- 3) $\delta_B(q, \perp, y_0) = \{(f, y_0)\}$; $q \in Q$.

Então:

$$x \in N(A) \Leftrightarrow (q_0, x, z_0) \vdash_A^* (q, \perp, \perp) \Leftrightarrow$$

$$\Leftrightarrow (q'_0, x, y_0) \vdash_B (q_0, x, z_0 y_0) \quad (\text{devido a (1)})$$

$$\vdash_B^* (q, \perp, y_0) \quad (\text{devido a (2)})$$

$$\vdash (f, \perp, y_0) \quad (\text{devido a (3)})$$

$$\Leftrightarrow x \in L(B).$$

Logo: $L(B) = N(A)$ e portanto $N_{\Sigma} \subseteq L_{\Sigma}$.

Então, dos lemas 3.4 e 3.5, fica provada a proposição 3.1, ou seja, a classe das linguagens aceitas por estado final e a classe das linguagens aceitas por pilha vazia são a mesma classe.

Proposição 3.2. $L_{\Sigma} = N_{\Sigma} = \{ L \subseteq \Sigma^* \mid L \text{ é linguagem livre de contexto} \}$

(ou seja, a classe de linguagens reconhecidas por autômatos de pilha e a classe de linguagens livres de contexto são a mesma classe)

Lema 3.6. Se L é linguagem livre de contexto, então $L = N(A)$ para algum autômato de pilha A .

Prova. Seja $G = (N, \Sigma, P, S)$ uma GLC tal que $L = L(G)$.

Construir $A = (\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \phi)$ com δ dada por:

$$(1) \quad \delta(q, \perp, B) = \{ (q, x) \mid B \rightarrow x \in P \}$$

$$(2) \quad \delta(q, a, a) = \{ (q, \perp) \} \quad ; \quad a \in \Sigma.$$

Será provado inicialmente que

$$\left[S \xRightarrow[q]{*} u\alpha \ ; \ u \in \Sigma^*, \alpha \in (N \cup \Sigma)^* \right] \Rightarrow$$

$$\Rightarrow \left[\forall w \in \Sigma^*, (q, uw, S) \vdash_A^* (q, w, \alpha) \right]$$

por indução no comprimento da derivação.

$$\text{base : } \left[S \xRightarrow[q]{0} S \right] \rightarrow \left[\forall w \in \Sigma^*, (q, w, S) \vdash^0 (q, w, S) \right]$$

(trivialmente)

hipótese indutiva : a proposição vale para derivações de comprimento n

passo da indução:

$$\text{Seja: } S \xRightarrow{n} u_1 \underbrace{A u_3}_{\alpha'} \Rightarrow \underbrace{u_1 u_2 u_3}_u \alpha$$

Então:

$$\forall w \in \Sigma^*, (q, u_1 u_2 u_3 w, S) \vdash^* (q, u_2 u_3 w, A u_3 \alpha) \quad (\text{hipótese})$$

$$\vdash (q, u_2 u_3 w, u_2 u_3 \alpha) \quad (\text{de (1)})$$

e de $A \rightarrow u_2 \in P$

$$\vdash^* (q, w, \alpha) \quad (\text{de (2)})$$

Particularizando este resultado para $\alpha = w = \perp$ tem-se:

$$\left[S \xRightarrow[q]{*} u ; u \in \Sigma^* \right] \Rightarrow \left[(q, u, S) \vdash_A^* (q, \perp, \perp) \right]$$

e portanto: $L(q) \subseteq N(A)$.

Exercício: Mostre por indução no número de transições que

$$\left[(q, uw, S) \vdash_A^* (q, w, \alpha) \right] \Rightarrow \left[S \xRightarrow[q]{*} u \alpha \right]$$

Particularizando este resultado para $w = \alpha = \perp$ tem-se que $N(A) \subseteq L(q)$.

Portanto: $L = L(q) = N(A)$.

EXEMPLO:

$$\begin{array}{l} E \rightarrow E+T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow x \mid (E) \end{array}$$

O reconhecedor para a linguagem gerada por essa gramática, pode ser construído utilizando-se o lema 3.6. Então:

$$A = (\{q\}, \Sigma, \Gamma, \delta, q, E, \phi) \text{ onde:}$$

$$\Sigma = \{x, +, *, (,)\} \quad ; \quad \Gamma = \Sigma \cup \{E, T, F\}$$

e δ dada por:

$$\delta(q, \perp, E) = \{(q, E+T), (q, T)\}$$

$$\delta(q, \perp, T) = \{(q, T * F), (q, F)\}$$

$$\delta(q, \perp, F) = \{(q, x), (q, (E))\}$$

$$\delta(q, a, a) = \{(q, \perp)\} \quad ; \quad a \in \Sigma.$$

Seja a expressão: $x+x*x$. Tem-se as transições de A:

$$\begin{aligned} & (q, x+x*x, E) \vdash (q, x+x*x, E+T) \vdash (q, x+x*x, T+T) \\ & \vdash (q, x+x*x, F+T) \vdash (q, x+x*x, x+T) \stackrel{2}{\vdash} (q, x*x, T) \\ & \vdash (q, x*x, T * F) \vdash (q, x*x, F * F) \vdash (q, x*x, x * F) \\ & \stackrel{2}{\vdash} (q, x, F) \vdash (q, x, x) \vdash (q, \perp, \perp). \end{aligned}$$

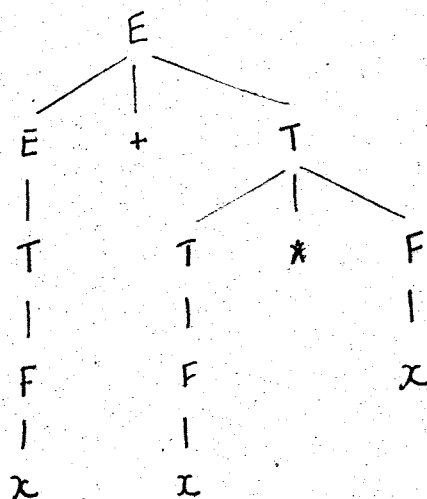
Observando as transições de A, nota-se que a fita de entrada contém a cada instante, a parte da cadeia de entrada que falta ser analisada e que o conteúdo da pilha simula uma derivação mais à esquerda:

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow x + T \Rightarrow x + T * F \Rightarrow$$

$$\Rightarrow x + F * F \Rightarrow x + x * F \Rightarrow x + x * x$$

Este autômato de pilha é conhecido como RECONHECEDOR DESCENDENTE ou ANALISADOR SINTÁTICO DESCENDENTE ("top-down parser") porque como simula derivações mais à esquerda, constrói a árvore sintática da cadeia de entrada, de cima para baixo.

Neste caso, tem-se:


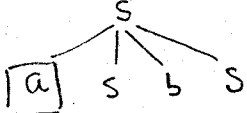
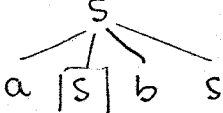
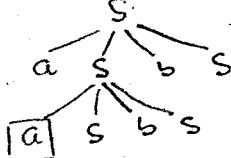
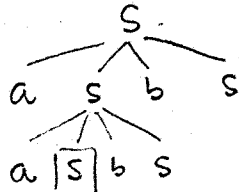
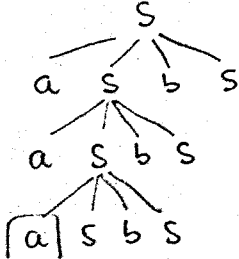
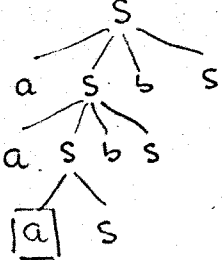
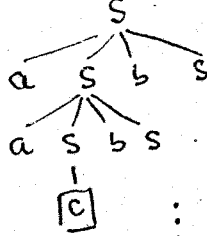


Entretanto, como pode ser observado da definição da função de transição de estado δ , o autômato A é não-determinístico, sendo que as transições mostradas foram "convenientemente" escolhidas a cada passo. Uma questão interessante é: "É possível automatizar este processo?". A resposta é sim. Um algoritmo, conhecido

Como ANÁLISE SINTÁTICA DESCENDENTE COM RETORNO ("top-down backtracking parsing"), é o seguinte:

- (1) começa a árvore sintática pelo símbolo inicial da pilha (que é o símbolo inicial da gramática). Esse símbolo é o nó ativo inicial. Executa em seguida, os passos (2) e (3) recursivamente.
- (2) se o nó ativo é um não-terminal A , escolhe a primeira A -produção ainda não utilizada, $A \rightarrow X_1 X_2 \dots X_k$ e cria na árvore sintática, X_1, X_2, \dots, X_k como descendentes diretos de A . Marca o primeiro destes descendentes (o mais à esquerda) como ativo.
- (3) se o nó ativo é um terminal a , então compará-lo com o símbolo atual de entrada (símbolo que está sendo lido na fila de entrada). Se eles forem iguais, então tornar ativo o símbolo imediatamente à direita de a na árvore e avançar com a cabeça leitora para o próximo símbolo de entrada. Se eles forem diferentes, retornar ao último não terminal A expandido para o qual existe uma A -produção ainda não utilizada e atualizar o símbolo de entrada.
- (4) se não existe nó ativo e todos os símbolos de entrada foram lidos, então a análise sintática termina com sucesso (a cadeia foi reconhecida). Caso contrário, rejeitar a cadeia (isto é, existe um erro sintático na cadeia).

EXEMPLO: Seja a gramática $S \rightarrow aSbS \mid aS \mid c$ e a cadeia de entrada aacbc.

ÁRVORE	CADENA	OBSERVAÇÕES
	\downarrow aacbc	O símbolo \square mostra o nó atualmente ativo; \downarrow indica o símbolo de entrada que está sendo lido
	\downarrow aacbc	escolhe-se a primeira alternativa para expandir S.
	a \downarrow acbc	o símbolo de entrada "casa" com o nó da árvore; o próximo símbolo é considerado.
	a \downarrow acbc	escolhe-se a primeira alternativa para expandir S
	aa \downarrow cbc	"casamento" de símbolos; passa ao próximo nó da árvore
	aa \downarrow cbc	o símbolo S foi expandido; A escolha no estado é incorreta porque não há "casamento" de terminais. Testa a próxima alternativa
	aa \downarrow cbc	essa alternativa também é incorreta; testa a próxima alternativa.
	aac \downarrow bc :	"casamento" :

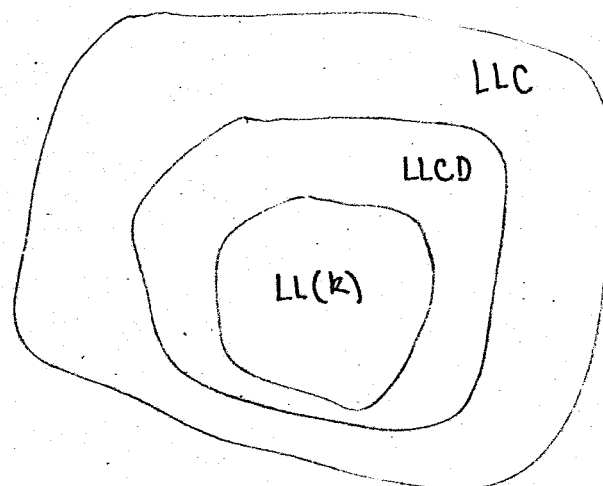
Para mais detalhes sobre esse algoritmo ver:

ATO & ULLMAN - "The theory of parsing, translation and compiling", vol. I, pg 289-291

Este algoritmo apresenta os seguintes inconvenientes:

- o número de passos necessário para uma análise pode ser muito grande (o algoritmo tem complexidade assintótica exponencial)
- a capacidade de indicar erros em cadeias de entrada mal formadas é muito pequena.

Existe um grande interesse, para a construção de compiladores, em analisadores sintáticos eficientes, ou seja, analisadores que evitam esses inconvenientes. A seguir serão estudadas, classes de gramáticas livres de contexto para as quais pode-se evitar o "backtracking" e construir reconhecedores eficientes (analisadores com complexidade linear no espaço e no tempo). Essas classes de gramáticas - gramáticas $LL(k)$ - no entanto, não geram todas as linguagens livres de contexto, apesar de existir forte evidência de que essas gramáticas sejam adequadas para especificar as características sintáticas de linguagens de programação. Esquemáticamente, tem-se:



onde:

LLC - classe das linguagens livres de contexto

LLCD - classe das linguagens livres de contexto determinísticas

$LL(k)$ - classe das linguagens geradas por gramáticas $LL(k)$.

Intuitivamente:

seja $G = (N, \Sigma, P, S)$ uma gramática não ambígua e $w = a_1 \dots a_n$ uma cadeia de $L(G)$. Existe então, uma única sequência de formas sentenciais obtidas por derivação mais à esquerda, $\alpha_0, \dots, \alpha_m$ tal que:

$$S = \alpha_0 \quad ; \quad \alpha_i \Rightarrow \alpha_{i+1} \quad , \quad 0 \leq i < m \quad ; \quad \alpha_m = w$$

Seja $\alpha_i = a_1 \dots a_j A \beta$. Se α_{i+1} poder ser determinada conhecendo-se apenas $a_1 \dots a_j$ (isto é, a parte da cadeia de entrada que já foi lida), A (isto é, o não terminal a ser substituído em seguida) e $a_{j+1} \dots a_{j+k}$ para algum inteiro k , então a gramática G é uma gramática $LL(k)$.

definição 3.14. Seja $G = (N, \Sigma, P, S)$ uma GLC. Defina-se o conjunto $FIRST_k(\alpha)$, de todos os prefixos de comprimento k (ou das cadeias de comprimento menor do que k) que podem ser derivados de α por:

$$FIRST_k(\alpha) = \{ x \in \Sigma^* / \alpha \xRightarrow{*} x\beta, |x| = k \text{ ou } \alpha \xRightarrow{*} x, |x| < k \}$$

definição 3.15. Seja $G = (N, \Sigma, P, S)$ uma GLC. Diz-se que G é $LL(k)$ para algum inteiro k , se sempre que existirem duas derivações mais à esquerda:

$$S \xRightarrow{*} wA\alpha \Rightarrow w\beta\alpha \xRightarrow{*} wx \quad e$$

$$S \xRightarrow{*} wA\alpha \Rightarrow w\gamma\alpha \xRightarrow{*} wy \quad \text{tais que } \text{FIRST}_L(x) = \text{FIRST}_L(y)$$

então $\beta = \gamma$.

EXEMPLO: Seja G com produções:

$$\begin{aligned} S &\rightarrow aAS \mid b \\ A &\rightarrow a \mid bSA \end{aligned}$$

Neste caso G é $LL(1)$ porque dado o não terminal mais a esquerda de uma forma sentencial (S ou A) e o próximo símbolo de entrada (a ou b , pois do contrário a cadeia é mal formada) só existe uma produção de G capaz de derivar um símbolo.

Exercício: mostrar (formalmente) que G é $LL(1)$!

definição 3.16. Uma GLC $G = (N, \Sigma, P, S)$ é $LL(1)$ simples se ela for Λ -livre e dado um par (A, a) ; $A \in N, a \in \Sigma$, existe no máximo uma produção $A \rightarrow a\alpha$. $\alpha \in (N \cup \Sigma)^*$.

EXEMPLOS:

(1) Seja G com produções:

$$\begin{aligned} S &\rightarrow \Lambda \mid abA \\ A &\rightarrow Saa \mid b \end{aligned}$$

Será mostrado que G é $LL(2)$. Seja:

$$S \xRightarrow{*} wS\alpha \Rightarrow w\beta\alpha \xRightarrow{*} wx$$

$$S \xRightarrow{*} wS\alpha \Rightarrow w\gamma\alpha \xRightarrow{*} wy$$

onde x e y começam com os mesmos 2 símbolos. As únicas duas maneiras de S aparecer numa forma sentencial são:

(a) $w = \alpha = \perp$ (ou seja, $S \xRightarrow{*} wS\alpha$ é, na verdade, $S \xRightarrow{0} S$)

(b) a última produção usada para se chegar na forma sentencial $wS\alpha$ foi $A \rightarrow Saa$

Logo: $\alpha = \perp$ ou α começa com aa .

Sejam as derivações:

$$\begin{aligned} wS\alpha &\Rightarrow w\beta\alpha \\ wS\alpha &\Rightarrow w\gamma\alpha \end{aligned}$$

Como x e y devem começar com os mesmos 2 símbolos, deve-se usar $S \rightarrow \perp$ ou $S \rightarrow abA$ em ambas as derivações, pois do contrário, como $\alpha = \perp$ ou α começa com aa , uma cadeia (por exemplo, x) seria \perp ou começaria com aa , ao passo que a outra (y , por exemplo) começaria com ab .

Logo: se for usada a produção $S \rightarrow \perp$ tem-se que $\beta = \gamma = \perp$ e se for usada a produção $S \rightarrow abA$ tem-se que $\beta = \gamma = abA$

Exercício: completar a prova, isto é, considerar o caso

$$S \xRightarrow{*} wA\alpha \Rightarrow w\beta\alpha \xRightarrow{*} wx$$

$$S \xRightarrow{*} wA\alpha \Rightarrow w\gamma\alpha \xRightarrow{*} wy$$

(2) Seja G com produções:

$$S \rightarrow A \mid B$$

$$A \rightarrow aAb \mid 0$$

$$B \rightarrow aBbb \mid 1$$

$$L(G) = \{a^n 0 b^n / n \geq 0\} \cup \{a^n 1 b^{2n} / n \geq 0\}$$

Será mostrado (informalmente) que G não é $LL(k)$, para qualquer k .

Ao analisar uma cadeia de a 's, não é possível saber qual a produção $S \rightarrow A$ ou $S \rightarrow B$ foi usada no início da derivação dessa cadeia, até que seja encontrado um 0 ou um 1. Logo, como a cadeia de a 's pode ser arbitrariamente grande, para qualquer inteiro k , é sempre possível escrever:

$$S \xRightarrow{0} S \Rightarrow A \xRightarrow{*} a^k 0 b^k$$

$$S \xRightarrow{0} S \Rightarrow B \xRightarrow{*} a^k 1 b^{2k}$$

Teorema 3.3. Seja $G = (N, \Sigma, P, S)$ uma GLC. G é $LL(k) \Leftrightarrow$ se $A \rightarrow \beta$ e $A \rightarrow \gamma$ são produções distintas de P então $\text{FIRST}_k(\beta\alpha) \cap \text{FIRST}_k(\gamma\alpha) = \emptyset$ para todo $wA\alpha$ tal que $S \xRightarrow{*} wA\alpha$.

Prova.

(\Leftarrow) Sejam $w, A, \alpha, \beta, \gamma$ como no enunciado. Seja x um elemento de $\text{FIRST}_k(\beta\alpha) \cap \text{FIRST}_k(\gamma\alpha)$. Logo, pela definição de FIRST_k tem-se:

$$S \xRightarrow{*} wA\alpha \Rightarrow w\beta\alpha \xRightarrow{*} wx\gamma$$

$$S \xRightarrow{*} wA\alpha \Rightarrow w\gamma\alpha \xRightarrow{*} wx\gamma$$

para algum $y \in Z$ (se $|x| < k$, então $y = z = \lambda$). Portanto, como $\beta \neq \gamma$, G não é $LL(k)$.

(\Rightarrow) EXERCÍCIO!

Seja $G = (N, \Sigma, P, S)$ uma GLC, λ -livre. Quer-se determinar se G é $LL(1)$. Pelo teorema 3.3 tem-se:

G é $LL(1) \Leftrightarrow$ para todo não terminal A , com A -produções $A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$,

$FIRST_1(\alpha_1), \dots, FIRST_1(\alpha_n)$ são disjuntos 2 a 2.

EXEMPLO: $S \rightarrow aS$
 $S \rightarrow a$ não é $LL(1)$, pois

$$FIRST_1(aS) = \{a\} = FIRST_1(a)$$

E se a gramática não for λ -livre?

definição 3.17. Seja $G = (N, \Sigma, P, S)$ uma GLC. Seja $\beta \in (N \cup \Sigma)^*$.

$$FOLLOW_k(\beta) = \{w \mid S \xRightarrow{*} \alpha\beta\gamma \text{ e } w \in FIRST_k(\gamma)\}$$

EXEMPLO: $FOLLOW_1(A)$ é o conjunto de símbolos terminais que podem aparecer imediatamente à direita de A numa forma sen-

lencial qualquer. Além disso se αA é uma forma sentencial possível, então $\alpha \in \text{Follow}_1(A)$.

Teorema 3.4. Uma GLC $G = (N, \Sigma, P, S)$ é LL(1) \Leftrightarrow para cada não terminal A , se $A \rightarrow \beta$ e $A \rightarrow \gamma$ são produções distintas, então

$$\text{FIRST}_1(\beta \text{Follow}_1(A)) \cap \text{FIRST}_1(\gamma \text{Follow}_1(A)) = \emptyset$$

Prova. Exercício!

definição 3.18. Uma GLC G tal que se $A \rightarrow \beta$ e $A \rightarrow \gamma$ são A -produções distintas, então

$$\text{FIRST}_k(\beta \text{Follow}_k(A)) \cap \text{FIRST}_k(\gamma \text{Follow}_k(A)) = \emptyset$$

é chamada gramática FORTEMENTE LL(k).

Ou seja, numa gramática fortemente LL(k), dadas duas derivações mais a esgruda

$$S \xRightarrow{*} w A \alpha_1 \Rightarrow w \beta \alpha_1 \xRightarrow{*} wx$$

$$S \xRightarrow{*} w A \alpha_2 \Rightarrow w \gamma \alpha_2 \xRightarrow{*} wy$$

tal que os primeiros k símbolos de x são iguais aos primeiros k símbolos de y , então $\beta = \gamma$.

Do leorema 3.4 re-x que toda gramática LL(1) é uma gramática fortemente LL(1). Entretanto, para $k > 1$, existem gramáticas LL(k) que não são fortemente LL(k).

Exemplo. Seja a gramática

$$\begin{aligned} S &\rightarrow aAaa \mid bAba \\ A &\rightarrow b \mid \perp \end{aligned}$$

(a) verificação se G é LL(1)

$$\begin{aligned} 1. \quad S &\rightarrow aAaa \\ S &\rightarrow bAba \end{aligned}$$

$$S \xRightarrow{0} S \quad (w = \alpha = \perp)$$

$$\text{FIRST}_1(\beta\alpha) = \text{FIRST}_1(aAaa) = \{a\}$$

$$\text{FIRST}_1(\gamma\alpha) = \text{FIRST}_1(bAba) = \{b\}$$

Logo $\text{FIRST}_1(\beta\alpha) \cap \text{FIRST}_1(\gamma\alpha) = \emptyset$ para as S-produções.

$$\begin{aligned} 2. \quad A &\rightarrow b \\ A &\rightarrow \perp \end{aligned}$$

Uma possível forma sentencial na qual A aparece é $bAba$ ($S \Rightarrow bAba$). Neste caso ($w = b$ e $\alpha = ba$) tem-se:

$$\text{FIRST}_1(\beta\alpha) = \text{FIRST}_1(bba) = \{b\}$$

$$\text{FIRST}_1(\gamma\alpha) = \text{FIRST}_1(ba) = \{b\}$$

Logo, $\text{FIRST}_1(\beta\alpha) \cap \text{FIRST}_1(\gamma\alpha) = \{b\} \neq \emptyset$ e portanto

G não é LL(1) (intuitivamente: estado A no topo da pilha e olhando b na cadeia de entrada, não é possível decidir se A deve ser expandido usando-se $A \rightarrow b$ ou $A \rightarrow \Lambda$).

(b) verificação se G é LL(2).

$$1. \text{FIRST}_2(aAaa) \cap \text{FIRST}_2(bAba) = \emptyset$$

$$2. \text{Para } S \xRightarrow{*} aAaa \quad (w=a; \alpha=aa)$$

$$\text{FIRST}_2(baa) \cap \text{FIRST}_2(aa) = \emptyset$$

$$\text{Para } S \xRightarrow{*} bAba \quad (w=b; \alpha=ba)$$

$$\text{FIRST}_2(bba) \cap \text{FIRST}_2(ba) = \emptyset$$

Logo, G é LL(2)!

(c) verificação se G é fortemente LL(2):

$$\text{FOLLOW}_2(A) = \{w \mid S \Rightarrow aAaa \text{ e } w \in \text{FIRST}_2(aa)$$

$$\text{ou } S \Rightarrow bAba \text{ e } w \in \text{FIRST}_2(ba)\} = \{aa, ba\}$$

Então: $\text{FIRST}_2(\beta \text{FOLLOW}_2(A)) \cap \text{FIRST}_2(\gamma \text{FOLLOW}_2(A))$ pode ser:

$$1. \text{FIRST}_2(baa) \cap \text{FIRST}_2(aa) = \emptyset$$

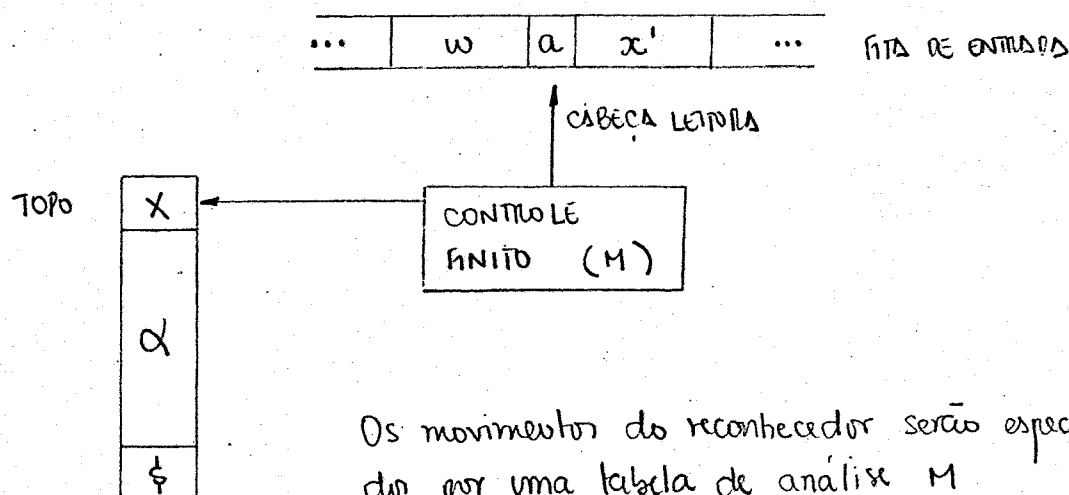
$$2. \text{FIRST}_2(bba) \cap \text{FIRST}_2(aa) = \emptyset$$

$$3. \text{first}_2(bba) \cap \text{first}_2(ba) = \emptyset$$

$$4. \text{first}_2(baa) \cap \text{first}_2(ba) = \{ba\}$$

e portanto, G não é fortemente $LL(2)$.

CONSTRUÇÃO DE RECONHECEDORES PARA LINGUAGENS $LL(1)$ (ANALISADORES SINTÁTICOS $LL(1)$)



$M: (N \cup \Sigma \cup \{\$, \top\}) \times (\Sigma \cup \{\perp\})$ definida por:

1. se $X \rightarrow \beta$ é a i -ésima produção em P , então $M(X, a) = (\beta, i)$ para todo $a \in \text{first}_1(\beta)$, $a \neq \perp$.
Se $\perp \in \text{first}_1(\beta)$ então $M(X, \perp) = (\beta, i)$ para todo $b \in \text{follow}_1(X)$
2. $M(a, a) = \text{pop}$ para todo $a \in \Sigma$.
3. $M(\$, \perp) = \text{sucesso}$
4. $M(X, a) = \text{erro}$, para os demais casos.

Num movimento, o símbolo a ser lido a e o símbolo no topo da pilha X são determinados. Então a entrada $M(X, a)$ na tabela de análise é consultada para determinar qual a transição a ser feita. Seja $a = \text{FIRST}_1(x)$.

- (1) se $M(X, a) = (\beta; i)$ então $(x, X\alpha) \vdash (x, \beta\alpha)$
- (2) se $M(a, a) = \text{POP}$ então $(x, a\alpha) \vdash (x', \alpha)$; $x = ax'$
- (3) $M(\$, _) = \text{SUCESSO}$ indica que $(_, \$)$ é configuração final
- (4) se $M(X, a) = \text{ERRO}$ então $(x, X\alpha)$ é configuração terminal.

EXEMPLO: Seja a gramática

- (1) $E \rightarrow TE'$
- (2) $E' \rightarrow +TE'$
- (3) $E' \rightarrow _$
- (4) $T \rightarrow FT'$
- (5) $T' \rightarrow *FT'$
- (6) $T' \rightarrow _$
- (7) $F \rightarrow (E)$
- (8) $F \rightarrow a$

Então:

$$\text{FIRST}_1(TE') = \{ (, a \}$$

$$\text{FIRST}_1(+TE') = \{ + \}$$

Como $E' \rightarrow _ \in P$:

$$\text{FOLLOW}_1(E') = \{ _,) \}$$

$$\text{FIRST}_1(FT') = \{ (, a \}$$

$$\text{FIRST}_1(*FT') = \{ * \}. \text{ Como } T' \rightarrow _ \in P : \text{FOLLOW}_1(T') = \{ _, +,) \}$$

$$\text{FIRST}_1((E)) = \{ (\}$$

$$\text{FIRST}_1(a) = \{ a \}$$

Logo, a tabela M para esta gramática será:

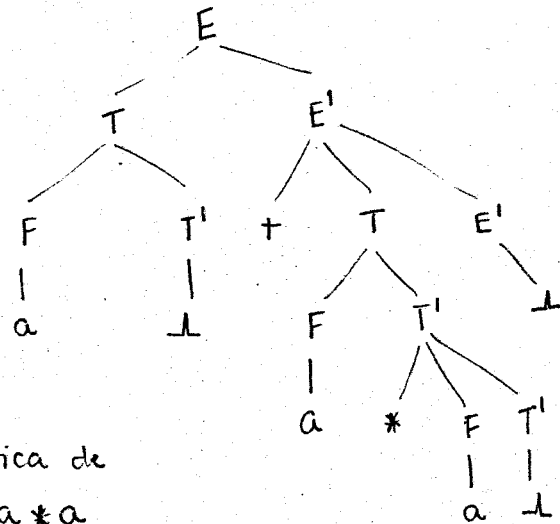
	+	*	()	a	⊥
E			TE', 1		TE', 1	
E'	+TE', 2			⊥, 3		⊥, 3
T			FT', 4		FT', 4	
T'	⊥, 6	*FT', 5		⊥, 6		⊥, 6
F			(E), 7		a, 8	
+	POP					
*		POP				
(POP			
)				POP		
a					POP	
⊥						SUCCESSO

Seja a cadeia $a + a * a$

ANÁLISE
SINTÁTICA

$(a + a * a, E' \S)$	$\vdash (a + a * a, TE' \S)$	1
	$\vdash (a + a * a, FT'E' \S)$	4
	$\vdash (a + a * a, aT'E' \S)$	8
	$\vdash (+a * a, T'E' \S)$	
	$\vdash (+a * a, E' \S)$	6
	$\vdash (+a * a, +TE' \S)$	2
	$\vdash (a * a, TE' \S)$	
	$\vdash (a * a, FT'E' \S)$	4
	$\vdash (a * a, aT'E' \S)$	8
	$\vdash (*a, T'E' \S)$	
	$\vdash (*a, *FT'E' \S)$	5
	$\vdash (a, FT'E' \S)$	
	$\vdash (a, aT'E' \S)$	8
	$\vdash (\perp, T'E' \S)$	
	$\vdash (\perp, E' \S)$	6
	$\vdash (\perp, \S)$	3
	SUCCESSO!	

Portanto tem-se:



Árvore sintática de
 $a+a*a$

Como se viu anteriormente, uma maneira de reconhecer linguagens livres de contexto é através de um analisador sintático descendente. Outra maneira é por meio da análise sintática ascendente ("bottom-up parsing"). Na análise sintática descendente tenta-se construir a árvore sintática da raiz (símbolo inicial da gramática) para as folhas (símbolos da cadeia de entrada). Na análise ascendente constrói-se a árvore das folhas para a raiz da seguinte maneira:

- considera-se a cadeia que está no topo da pilha e verifica-se se existe uma produção cujo lado direito corresponde a esse símbolo. Se existir, troca-se os símbolos do topo da pilha, que serão denominados LEQUE ("handle"), pelo lado esquerdo da produção (essa operação é denominada REDUÇÃO)
- se nenhuma redução é possível, EMPILHA-SE o próximo símbolo da cadeia de entrada, move-se a cabeça leitora para o próximo símbolo de entrada e o processo é reiniciado.

- ao se chegar ao fim da cadeia sem que nenhuma redução seja possível, retrorna-se à configuração (isto é, posição da cabeça leitora e conteúdo da pilha) anterior à última redução e tenta-se outra alternativa.

EXEMPLO: Seja a gramática:

- (1) $S \rightarrow AB$
- (2) $A \rightarrow ab$
- (3) $B \rightarrow aba$

e a cadeia de entrada: ababa. Tem-se então:

FITA DE ENTRADA	PILHA	OPERACAO	ÁRVORE
↓ ababa		EMPILHA	
a↓ baba	a	EMPILHA	a
ab↓ aba	ab	REDUZ COM (2)	a b
ab↓ aba	A	EMPILHA	<pre> A / \ a b </pre>
ab↓ aba	Aa	EMPILHA	<pre> A / \ a b / \ a b a </pre>
abab↓ a	Aab	REDUZ COM (2)	<pre> A / \ a b / \ a b a b </pre>
abab↓ a	AA	EMPILHA	<pre> A A / \ / \ a b a b / \ a b a b </pre>
ababa↓	AAa		<pre> A A / \ / \ a b a b / \ a b a b a </pre>

neste ponto chega-se ao fim da cadeia de entrada e nenhuma redução é possível. Deve-se voltar à configuração anterior à última redução.

abab↓ a	Aab	<pre> A / \ a b / \ a b a b </pre>
---------	-----	---

como não há alternativa para redução, tenta-se o empilhamento:

ababa [↑]	Aaba	REDUZ com (3)	<pre> A / \ a b a b a </pre>
ababa [↓]	AB	REDUZ com (1)	<pre> A B / \ / \ a b a b a </pre>
ababa [↓]	S		<pre> S / \ A B / \ / \ a b a b a </pre>

Neste caso, como se chegou ao fim da cadeia de entrada, com o símbolo inicial da gramática no topo da pilha, a análise sintática termina com sucesso. O reverso da sequência de produções usadas nas reduções irá corresponder a uma derivação mais à direita da cadeia de entrada:

$$S \Rightarrow AB \Rightarrow Aaba \Rightarrow ababa$$

Este método, conhecido como ANÁLISE SINTÁTICA ASCENDENTE COM RETORNO ("bottom-up backtrack parsing") considera todos os possíveis movimentos de um autômato de pilha não determinístico para a gramática e é portanto, muito ineficiente. Além disso, \perp -produções causam dificuldade pois, pode-se fazer um número arbitrário de reduções nas quais \perp é "reduzida" a um não terminal. Para mais detalhes sobre esse algoritmo ver:

ALTO & ULLMAN - "The theory of parsing, translation and compiling", vol. I, pg 303-304

A seguir será mostrado uma classe de gramáticas livres de contexto - denominadas gramáticas LR(k) - para as quais pode-se construir analisadores sintáticos ascendentes determinísticos e portan-

to, eficientes.

Existem 2 decisões que um analisador ascendente determinístico deve tomar:

- determinar, em cada movimento, se empilha um símbolo de entrada ou se faz uma redução (isto é equivalente a determinar a extremidade direita de um leque);
- uma vez determinado o leque na pilha (isto é, determinado sua extremidade esquerda) determinar qual produção usar na redução.

Invariavelmente, uma GLC é LR(k) se dada uma derivação mais à direita:

$$S \Rightarrow \alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_m = z$$

onde $\alpha_{i-1} = \alpha A w$, $\alpha_i = \alpha \beta w$ (ou seja, β é o leque de α_i)

e $\beta = x_1 x_2 \dots x_\ell$, então:

- conhecendo-se $\alpha x_1 \dots x_j$ e os primeiros k símbolos de $x_{j+1} \dots x_\ell w$, pode-se estar certo de que o extremo direito do leque só é alcançado quando $j = \ell$
- conhecendo-se $\alpha \beta$ e no máximo os k primeiros símbolos de w , sempre é possível determinar que β é o leque e que β deve ser reduzido para A .

definição 3.19. Seja $G = (N, \Sigma, P, S)$ uma GLC. G é LR(k), $k \geq 0$ se sempre que:

- (1) $S \xRightarrow{*} \alpha A w \Rightarrow \alpha \beta w$
- (2) $S \xRightarrow{*} \gamma B x \Rightarrow \alpha \beta y$

$$(3) \text{FIRST}_k(w) = \text{FIRST}_k(y)$$

$$\text{então: } \alpha A y = \gamma B x \quad (\text{ou seja: } \alpha = \gamma; A = B; y = x)$$

Inintuitivamente, essa definição diz que se $\alpha\beta w$ e $\alpha\beta y$ são formas sentenciais de G , se $\text{FIRST}_k(w) = \text{FIRST}_k(y)$ e se $A \rightarrow \beta$ foi a última produção usada para obter $\alpha\beta w$ numa derivação mais à direita, então $A \rightarrow \beta$ deve também ser usada para reduzir $\alpha\beta y$.

A seguir é discutido, informalmente, o funcionamento de um algoritmo de análise sintática para gramáticas LR(k). Os movimentos de um analisador LR(k) são especificados por tabelas que contêm todas as informações necessárias à análise, ou seja:

- empilhar ou reduzir?
- que produção usar numa redução?

Uma das tabelas é denominada TABELA INICIAL e cada tabela LR(k) consiste de duas funções:

(1) FUNÇÃO DE ANÁLISE

$$f: \Sigma^{*k} \rightarrow \{\text{EMPILHA, REDUZ COM } i, \text{ERRO, SUCESSO}\}$$

(2) FUNÇÃO DE MOVIMENTO

$$g: (N \cup \Sigma) \rightarrow (\mathbb{T} \cup \{\text{ERRO}\})$$

onde \mathbb{T} é o conjunto de todas as tabelas LR(k).

A ideia da função de movimento g é evitar que a pilha precise ser analisada a cada passo para saber quando o leque aparece no topo (os valores de g serão como estados de um autômato reconhecedor de leques;

os estados do autômato serão empilhados também e o estado no topo da pilha corresponde ao estado que o autômato estaria se tivesse lido a pilha de baixo para cima)

ALGORÍTIMO DE ANÁLISE

Seja (w, T_0) a configuração inicial do analisador, onde w é a cadeia de entrada (conteúdo da fita de entrada) e T_0 é a tabela LR(k) inicial (topo da pilha)

(1) determinar a cadeia a ser lida \underline{u} , constituída pelos próximos \underline{k} símbolos de entrada

(2) determinar $f_T(u)$ para a tabela T do topo da pilha

(a) se $f_T(u) = \text{EMPILHA}$, então $(\alpha\beta, \alpha) \vdash (\beta, \alpha g_T(\alpha))$

(b) se $f_T(u) = \text{REDUZ}$ com i e a i -ésima produção é $A \rightarrow \delta$, então $(\gamma, \alpha\beta) \vdash (\gamma, \alpha X A g_X(A))$, onde $X \in \overline{T}$ e $|\beta| = 2|\delta|$ (ou seja, são removidos da pilha, o leque e todos os valores de g associados a ele)

(c) se $f_T(u) = \text{ERRO}$, então terminar

(d) se $f_T(u) = \text{SUCESSO}$, então o reverso da seqüência de números de produções usadas nas reduções do passo (b), é a análise sintática ascendente de w (ou equivalentemente, é a seqüência de derivações mais à direita de w)

EXEMPLO : Seja a gramática LR(1) :

- (1) $A \rightarrow AaAb$
- (2) $A \rightarrow \lambda$

e tabelas :

	f			g		
	a	b	λ	A	a	b
T_0	2		2	T_1		
T_1	E		Su		T_2	
T_2	2	2		T_3		
T_3	E	E			T_4	T_5
T_4	2	2		T_6		
T_5	2		1			
T_6	E	E			T_4	T_7
T_7	1	1				

onde : $E \equiv$ empilha ; número $i \equiv$ reduz usando produção i
 $Su \equiv$ sucesso e as entradas em branco correspondem a configurações de erro.

Seja a cadeia de entrada $w = aabb$

$$(aabb, T_0) \vdash (aabb, T_0 A T_1)$$

$$\text{pois : } f_{T_0}(a) = 2 ;$$

$$(2) A \rightarrow \lambda$$

$$g_{T_0}(A) = T_1$$

$$\vdash (abb, T_0 A T_1 a T_2)$$

$$\text{pois } f_{T_1}(a) = \text{EMPILHA}$$

$$g_{T_1}(a) = T_2$$

$$\vdash (abb, T_0 A T_1 a T_2 A T_3)$$

$$\vdash (bb, T_0 A T_1 a T_2 A T_3 a T_4)$$

$$\vdash (bb, T_0 A T_1 a T_2 A T_3 a T_4 A T_6)$$

$$\vdash (b, T_0 A T_1 a T_2 A T_3 a T_4 A T_6 b T_7)$$

$$\vdash (b, T_0 A T_1 a T_2 A T_3)'$$

$$\vdash (\perp, T_0 A T_1 a T_2 A T_3 b T_5)$$

$$\vdash (\perp, T_0 A T_1)$$

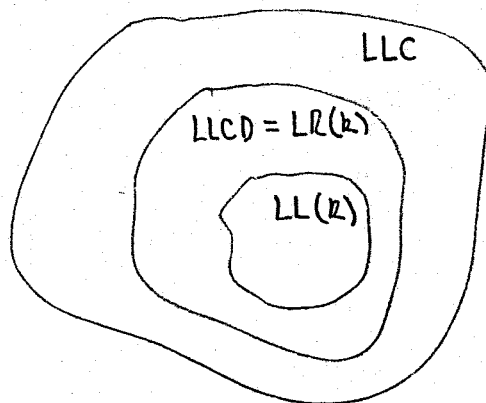
$$\vdash \text{SUCESSO}$$

Neste caso a sequência de produções usadas nas reduções é: 22211
Portanto, 11222 é a análise sintática ascendente de aabb.
A sequência de derivações mais à direita é:

$$\begin{array}{ccccccc} A & \Rightarrow & AaAb & \Rightarrow & AaAaAb & \Rightarrow & AaAabb \Rightarrow Aaabb \Rightarrow \\ (1) & & (1) & & (2) & & (2) \end{array}$$

$$\begin{array}{l} \Rightarrow aabb. \\ (2) \end{array}$$

Como foi mostrado anteriormente (ver diagrama na pg 87), a classe das linguagens geradas por gramáticas $LL(k)$ (no diagrama representada por $LL(k)$) é um subconjunto próprio das linguagens livres de contexto determinísticas. A classe de linguagens geradas por gramáticas $LR(k)$, no entanto, é a mesma classe das linguagens livres de contexto determinísticas, ou seja:



Para mais detalhes sobre gramáticas $LR(k)$ e analisadores sintáticos ascendentes determinísticos, ver

ALTO & ULLMAN - "Principles of compiler design", pg 197-244

ALTO & ULLMAN - "The theory of parsing, translation and compiling"
vol I, pg 368-396.