

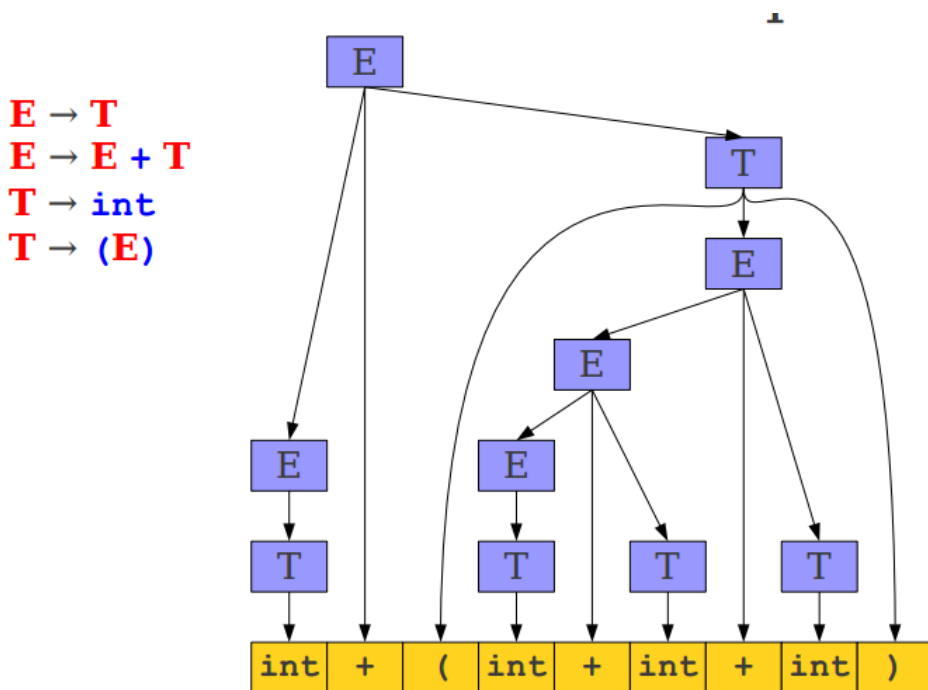
METODA DE ANALIZĂ SINTACTICĂ BOTTOM-UP (ASCENDENTĂ)

În cazul acestei metode, pentru o gramatică independentă de context dată se pornește de la șirul analizat și se realizează reduceri succesive până când (eventual) se ajunge la simbolul de start al gramaticii.

Din punctul de vedere al compilatoarelor, algoritmul general bottom-up (care utilizează backtracking) este irelevant.

Vom aplica metoda bottom-up într-un mod determinist pentru o clasă de gramatici independente de context, gramaticile de tip LR , pentru care se pot construi algoritmi de analiză sintactică liniari.

EXEMPLU



$$\begin{array}{ll}
\mathbf{E} \rightarrow \mathbf{T} & \text{int + (int + int + int)} \\
\mathbf{E} \rightarrow \mathbf{E} + \mathbf{T} & \Rightarrow \mathbf{T} + (\text{int + int + int}) \\
\mathbf{T} \rightarrow \text{int} & \Rightarrow \mathbf{E} + (\text{int + int + int}) \\
\mathbf{T} \rightarrow (\mathbf{E}) & \Rightarrow \mathbf{E} + (\mathbf{T} + \text{int + int}) \\
& \Rightarrow \mathbf{E} + (\mathbf{E} + \text{int + int}) \\
& \Rightarrow \mathbf{E} + (\mathbf{E} + \mathbf{T} + \text{int}) \\
& \Rightarrow \mathbf{E} + (\mathbf{E} + \text{int}) \\
& \Rightarrow \mathbf{E} + (\mathbf{E} + \mathbf{T}) \\
& \Rightarrow \mathbf{E} + (\mathbf{E}) \\
& \Rightarrow \mathbf{E} + \mathbf{T} \\
& \Rightarrow \mathbf{E}
\end{array}$$

GRAMATICI ȘI LIMBAJE DE TIP *LR*

Gramaticile de tip $LR(k)$ sunt gramatici neambigue, pentru care se pot implementa algoritmi de parsare liniari de tip bottom-up. Denumirea acestora provine din *Parsing from Left to Right using Rightmost derivations and k symbols lookahead*. Intuitiv, o gramatică este de tip $LR(k)$ dacă atunci când se ajunge într-un punct în care algoritmul trebuie să ia o decizie în privința reducerii care urmează să se aplice pentru un sufix (unic recunoscut) al formei sentențiale curente, această decizie poate fi luată în mod determinist (unic) analizând cel mult k simboluri înainte din intrarea curentă (acestea se numesc simboluri *lookahead*). Formal, avem următoarea

Definiție. Fie $G = (N, \Sigma, S, P)$ o gramatică independentă de context și $k \geq 0$ dat. Spunem că G este de tip $LR(k)$ dacă pentru orice două derivări drepte:

$$S \xRightarrow[d]{*} \alpha A u \xRightarrow[d]{*} \alpha \beta u = \gamma u, u \in \Sigma^* \quad \text{și} \quad S \xRightarrow[d]{*} \alpha' A' u' \xRightarrow[d]{*} \alpha' \beta' u' = \alpha \beta v = \gamma v, v \in \Sigma^*$$

astfel încât $FIRST_k(u) = FIRST_k(v)$, **atunci** $\alpha = \alpha', A = A', \beta = \beta'$.

Definiție. Spunem că gramatica G este de tip LR dacă există $k \geq 0$ pentru care G este de tip $LR(k)$.

Spunem că un limbaj L este de tip $LR(k)$ dacă există o gramatică independentă de context de tip $LR(k)$ care îl generează.

Spunem că L este de tip LR dacă există o gramatică independentă de context de tip LR care îl generează.

Propoziția 1. Orice gramatică de tip $LR(k)$ este neambiguă.

Demonstrație. Fie $G = (N, \Sigma, S, P)$ o gramatică independentă de context de tip $LR(k)$. Presupunem că G este ambiguă. Atunci există $z \in \Sigma^*, z \in L(G)$, care are două derivări drepte distincte de forma:

$$S \xRightarrow{*}_d \alpha Au \xRightarrow{*}_d \alpha \beta u \xRightarrow{*}_d z \quad \text{și} \quad S \xRightarrow{*}_d \alpha' A' u' \xRightarrow{*}_d \alpha' \beta' u' = \alpha \beta u \xRightarrow{*}_d z$$

unde $\alpha \beta u \xRightarrow{*}_d z$ este porțiunea ultimă comună a celor 2 derivări, $A \rightarrow \beta \neq A' \rightarrow \beta'$

sunt două producții distincte. Dar, deoarece G este $LR(k)$ și evident că $First_k(u) = First_k(u)$, rezultă că $\alpha = \alpha', A = A', \beta = \beta'$ adică $A \rightarrow \beta = A' \rightarrow \beta'$, contradicție.

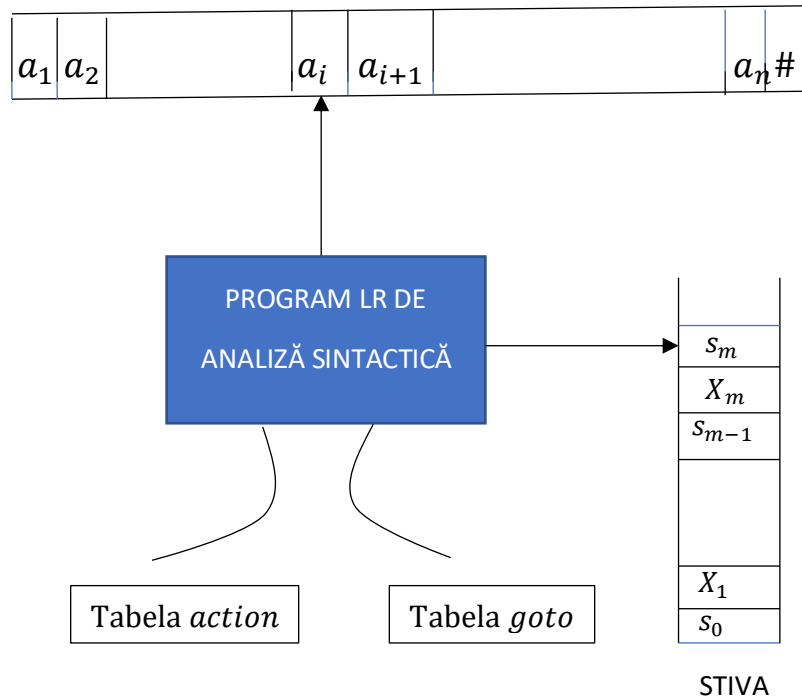
PARSERE DE TIP LR

- folosesc metoda bottom-up
- sunt liniare, au la bază un translator stivă determinist
- metoda de analiză LR este o metodă de tip deplasare-reducere, ușor de implementat și eficientă
- un analizor sintactic de tip LR poate detecta rapid o eroare de sintaxă, permițând revenirea din eroare
- determinarea tabelului de analiză sintactică este destul de complexă (acesta este singurul dezavantaj), dar există generatoare de analizoare sintactice LR precum *YACC* sau *bison*

ALGORITMUL LR

Fie $G = (N, \Sigma, S, P)$ gramatică independentă de context și extensia sa definită prin $G' = (N \cup \{S'\}, \Sigma \cup \{\#\}, S', P \cup \{S' \rightarrow S\})$ cu $L(G) = L(G')$.

Algoritmul sintactic de tip LR pentru G se bazează pe o tabelă de analiză sintactică constând din 2 părți, *action* și *goto*.



Programul de analiză sintactică LR este același pentru toate tipurile de gramatici de tip $LR(1)$, $SLR(1)$, $LALR(1)$, $LR(0)$ (pe care le vom studia), diferă doar modul de obținere al tabelii de analiză sintactică (tabelele *action* și *goto*). De fapt, acest algoritm are la bază un translator stivă determinist.

Programul citește caracterele șirului de intrare, unul câte unul. Este utilizată o stivă în care este păstrat un șir de forma $s_0 X_1 s_1 \dots X_m s_m$, unde $X_i \in N \cup \Sigma$, s_0, s_1, \dots, s_m sunt stările unui AFD care recunoaște mulțimea prefixelor viabile ale lui G , s_0 fiind starea inițială a acestui automat.

Simbolul din vârful stivei (care este întotdeauna o stare) și simbolul curent din intrare sunt folosite pentru determinarea acțiunii (deplasare sau reducere), pe baza tabelilor *action* și *goto*.

Programul funcționează astfel:

- determină simbolul din vârful stivei, s_m , și simbolul curent din intrare, a_i ;
- consultă $action[s_m, a_i]$ care poate fi:
 - i) *shift* s, s stare (deplasare)
 - ii) *reduce* $A \rightarrow \beta$
 - iii) *accept*
 - iv) *error*
- Funcția *goto* are ca argumente o stare și un neterminal și produce o stare
- O configurație a unui parser *LR* este de forma
$$(s_0 X_1 s_1 \dots, X_m s_m, a_i a_{i+1} \dots a_n \#, \pi)$$
unde $X_1 \dots X_m a_i \dots a_n$ este o formă sentențială a lui G dacă $a_1 \dots a_n \in L(G)$ și π este o derivare dreaptă parțială a sa.
- Configurația inițială este $(s_0, a_1 \dots a_n \#, \lambda)$, $a_1 \dots a_n \in \Sigma^*$ șirul analizat, s_0 starea inițială a AFD care recunoaște mulțimea prefixelor viabile.

Mișcările parserului:

- 1) $(s_0 X_1 s_1 \dots X_m s_m, a_i a_{i+1} \dots a_n \#, \pi) \vdash (s_0 X_1 s_1 \dots X_m s_m a_i s, a_{i+1} \dots a_n \#, \pi)$, dacă $action[s_m, a_i] = shift\ s$
- 2) $(s_0 X_1 s_1 \dots s_{j-1} X_j s_j \dots X_m s_m, a_i \dots a_n \#, \pi) \vdash (s_0 X_1 s_1 \dots s_{j-1} A s, a_i \dots a_n \#, k\pi)$, dacă $action[s_m, a_i] = reduce\ A \rightarrow X_j \dots X_m$ și $goto[s_{j-1}, A] = s$, iar k este numărul de ordine al producției $A \rightarrow X_j \dots X_m$, $1 \leq k \leq |P|$. Dacă $goto[s_{j-1}, A] = error$, șirul este respins. STOP.
- 3) dacă $action[s_m, a_i] = accept$, atunci $a_1 \dots a_n \in L(G)$ și $\pi \in \{1, \dots, |P|\}^*$ reprezintă derivarea sa dreaptă. STOP.
- 4) dacă $action[s_m, a_i] = error$, $a_1 \dots a_n \notin L(G)$. STOP.

Algoritmul, în pseudocod, este:

INPUT. Gramatica independentă de context $G = (N, \Sigma, S, P)$; G' extensia lui G ,
tabelele *action*, *goto*; w şirul analizat, $|w| = n, w = w_1 \dots w_n, w_{n+1} = \#$.

Configuraţia iniţială este: $(s_0, w\#, \lambda)$

OUTPUT. O derivare dreaptă (unică) a lui w , dacă $w \in L(G)$; un mesaj de eroare
dacă $w \notin L(G)$.

$Q \leftarrow \emptyset, S \leftarrow s_0$ // Q coadă, S stiva initializata cu s_0

$i \leftarrow 1$ // i pointer către primul caracter al lui w (sau primul *token*)

while(*true*){

$s \leftarrow$ simbolul din varful stivei;

$a \leftarrow a_i$;

if(*action*[s, a] == *shift* s'){

push a ; //deplaseaza a pe stiva

push s' ;

$i \leftarrow i + 1$; //avanseaza la urmatorul simbol (token) al lui w

 }

else if(*action*[s, a] == *reduce* $k: A \rightarrow \beta$){

pop $2 * |\beta|$ simboluri din stiva; //daca $\beta = \lambda$ nu se scoate nimic

$s' \leftarrow$ simbolul din varful stivei;

push A ;

if (*goto*[s', A] $\neq \emptyset$)

push *goto*[s', A]

else

error();

$k \Rightarrow Q$;

 }

else if(*action*[s, a] == "*accept*")

accept() // returneaza derivarea dreapta, aflata in Q

else

error()

} // end_while

PARSER LR PENTRU GRAMATICI LR(1)

Fie $G = (N, \Sigma, S, P)$ o gramatică independentă de context și G' extensia sa.

Definiție. Spunem că $\gamma \in (N \cup \Sigma)^*$ este un prefix viabil pentru G dacă în G există derivarea $S \xRightarrow{*}_d \alpha A w \xRightarrow{*}_d \alpha \beta w, \alpha, \beta \in (N \cup \Sigma)^*, w \in \Sigma^*$ și γ este prefix al lui $\alpha \beta$.

Definiție. Numim configurație LR(1) pentru G' perechea $[A \rightarrow \alpha. \beta; a]$, unde $A \rightarrow \alpha \beta \in P, a \in \Sigma \cup \{\#\}$. O configurație de forma $[A \rightarrow \alpha.; a], a \in Follow(A) (\# \in Follow(S))$, se numește finală.

Definiție. Spunem că $[A \rightarrow \alpha. \beta; a]$ este validă pentru prefixul viabil γ dacă în G avem:

- a) $\exists S \xRightarrow{*}_d \delta A w \xRightarrow{*}_d \delta \alpha \beta w, w \in \Sigma^*$
- b) $\gamma = \delta \alpha$
- c) $a = First(w\#)$ (dacă $w = \lambda$, atunci $a = \#$)

Determinarea mulțimilor canonice LR(1) pentru gramatica G (G')

Construim trei funcții: *closure*, *goto*, *config*. *closure* se calculează pentru o mulțime de configurații LR(1) și reprezintă mulțimea tuturor configurațiilor ce se obțin plecând de la configurații care au $'.'$ în fața unui neterminal. În acest caz se adaugă toate producțiile acelui neterminal cu $'.'$ în fața membrului drept, pentru a permite parsarea întregului membru drept al producției nou introduse.

- $closure(I) // I$ mulțime de configurații LR(1)
 - {
 - $J \leftarrow I;$
 - do{
 - for(fiecare $A \rightarrow \alpha. B \beta; a \in J$)
 - for(fiecare $B \rightarrow \gamma \in P$)
 - for(fiecare $b \in First(\beta a)$
 - if ($B \rightarrow \gamma; b \notin J$)
 - adauga $B \rightarrow \gamma; b$ la J
 - }
 - }
 - }
 - }

```

        }while ( se adauga noi configuratii la J )
    return J;
} // end_closure (I)

```

- ```

goto(I,X) // I mulțime de configurații LR(1), X ∈ N ∪ Σ
{
 J ← {A → αX.β; a | A → α.Xβ; a ∈ I}
 return(closure(J));
} // end_goto

```
- ```

config(G) // returneaza multimea configuratiilor canonice C_G
{
    C_G ← {closure({S' →.S; #})} cu closure({S' →.S; #}) nemarcata
    while( exista I ∈ C_G nemarcata )
    {
        marcheaza I;
        for( fiecare X ∈ N ∪ Σ )
            if( goto(I,X) ≠ ∅ && goto(I,X) ∉ C_G )
                adauga goto(I,X) la C_G ca stare nemarcata;
    } //end_while
    return C_G;
} // end_config

```

Observație. Vom arăta ulterior că mulțimile din C_G constituie stările unui AFD care recunoaște mulțimea prefixelor viabile ale lui G (G').

Tabela $LR(1)(action, goto)$ pentru gramatica G

INPUT: $G = (N, \Sigma, S, P)$ o gramatică independentă de context și G' extensia sa

OUTPUT: tabelele *action*, *goto* asociate lui G (G')

ALGORITMUL:

1. Calculeaza $C_G = \{I_0, I_1, \dots, I_n\}$, unde $I_0 = closure(\{S' \rightarrow .S; \#\})$
2. Pentru fiecare $j, 0 \leq j \leq n$, j stare ce corespunde lui I_j
 - 2.1. Daca $[A \rightarrow \alpha.a\beta; b] \in I_j$ && $goto(I_j, a) = I_k, a \in \Sigma$, atunci
 $action[j, a] = shift\ k$

2.2. Dacă $[A \rightarrow \alpha.; a] \in I_j, A \neq S'$, atunci

$$action[j, a] = reduce\ A \rightarrow \alpha$$

2.3. Dacă $[S' \rightarrow S.; \#] \in I_j$,

$$action[j, \#] = accept$$

2.4. $\forall x \in \Sigma \cup \{\#\}, action[j, x] = error$, în rest

2.5. Pentru fiecare $A \in N$

2.5.1. Dacă $goto(I_j, A) = I_k$, atunci

$$goto[j, A] = k$$

2.5.2. Dacă $goto(I_j, A) = \emptyset$, atunci

$$goto[j, A] = error$$

Observație. Vom arăta ulterior că *action* nu are intrări multiple dacă și numai dacă G este $LR(1)$.

EXEMPLU DE GRAMATICĂ $LR(1)$

Fie G_1 gramatica cu producțiile:

$$1: S \rightarrow L = R$$

$$2: S \rightarrow R$$

$$3: L \rightarrow * R$$

$$4: L \rightarrow a$$

$$5: R \rightarrow L$$

Mulțimile canonice $LR(1)$ pentru G_1 sunt:

$$I_0 \left[\begin{array}{l} S' \rightarrow .S; \# \\ S \rightarrow .L = R; \# \\ S \rightarrow .R; \# \\ L \rightarrow .* R; = \\ L \rightarrow .a; = \\ R \rightarrow .L; \# \\ L \rightarrow .* R; \# \\ L \rightarrow .a; \# \end{array} \right] = \left[\begin{array}{l} S' \rightarrow .S; \# \\ S \rightarrow .L = R; \# \\ S \rightarrow .R; \# \\ L \rightarrow .* R; = | \# \\ L \rightarrow .a; = | \# \\ R \rightarrow .L; \# \end{array} \begin{array}{l} \xrightarrow{goto(I_0, S)} I_1 \\ \xrightarrow{goto(I_0, L)} I_2 \\ \xrightarrow{goto(I_0, R)} I_3 \\ \xrightarrow{goto(I_0, *)} I_4 \\ \xrightarrow{goto(I_0, a)} I_5 \\ \xrightarrow{goto(I_0, L)} I_2 \end{array} \right]$$

$$I_1 = [S' \rightarrow S.; \#]$$

$$I_3 = [S \rightarrow R.; \#]$$

$$I_2 = \left[\begin{array}{l} S \rightarrow L. = R; \# \rightarrow I_6 \\ R \rightarrow L.; \# \end{array} \right]$$

$$I_4 = \begin{bmatrix} L \rightarrow *.R; = | \# \rightarrow I_7 \\ R \rightarrow .L; = | \# \rightarrow I_8 \\ L \rightarrow .*R; = | \# \rightarrow I_4 \\ L \rightarrow .a; = | \# \rightarrow I_5 \end{bmatrix}$$

$$I_5 = [L \rightarrow a.; = | \#]$$

$$I_6 = \begin{bmatrix} S \rightarrow L = .R; \# \rightarrow I_9 \\ R \rightarrow .L; \# \rightarrow I_{10} \\ L \rightarrow .*R; \# \rightarrow I_{11} \\ L \rightarrow .a; \# \rightarrow I_{12} \end{bmatrix}$$

$$I_7 = [L \rightarrow *R.; = | \#]$$

$$I_8 = [R \rightarrow L.; = | \#]$$

$$I_9 = [S \rightarrow L = R.; \#]$$

$$I_{10} = [R \rightarrow L.; \#]$$

$$I_{11} = \begin{bmatrix} L \rightarrow *.R; \# \rightarrow I_{13} \\ R \rightarrow .L; \# \rightarrow I_{10} \\ L \rightarrow .*R; \# \rightarrow I_{11} \\ L \rightarrow .a; \# \rightarrow I_{12} \end{bmatrix}$$

$$I_{12} = [L \rightarrow a.; \#]$$

$$I_{13} = [L \rightarrow *R.; \#]$$

Tabela de analiză sintactică $LR(1)$ pentru G_1 :

	<i>action</i>				<i>goto</i>		
	=	a	*	#	S	L	R
0	error	shift 5	shift 4	error	1	2	3
1	error	error	error	accept	error	error	error
2	shift 6	error	error	reduce 5	error	error	error
3	error	error	error	reduce 2	error	error	error
4	error	shift 5	shift 4	error	error	8	7
5	reduce 4	error	error	reduce 4	error	error	error
6	error	shift 12	shift 11	error	error	10	9
7	reduce 3	error	error	reduce 3	error	error	error
8	reduce 5	error	error	reduce 5	error	error	error
9	error	error	error	reduce 1	error	error	error
10	error	error	error	reduce 5	error	error	error
11	error	shift 12	shift 11	error	error	10	13
12	error	error	error	reduce 4	error	error	error
13	error	Error	error	reduce 3	error	error	error

Analizăm șirul " $* a = a$ ":

$$(0, * a = a\#, \lambda) \xrightarrow{S_4} (0 * 4, a = a\#, \lambda) \xrightarrow{S_5} (0 * 4 \underline{a5}, = a\#, \lambda) \xrightarrow{r_4} (0 * 4 \underline{L8}, = a\#, 4)$$

$$\xrightarrow{r_5} (0 * \underline{4R7}, = a\#, 54) \xrightarrow{r_3} (0L2, = a\#, 354) \xrightarrow{S_6} (0L2 = 6, a\#, 354)$$

$$\xrightarrow{S_{12}} (0L2 = 6 \underline{a12}, \#, 354) \xrightarrow{r_4} (0L2 = 6 \underline{L10}, \#, 4354) \xrightarrow{r_5} (0 \underline{L2 = 6R9}, \#, 54354)$$

$$\xrightarrow{r_1} (0S1, \#, 154354) \vdash \text{accept}$$

Șirul 154354 reprezintă derivarea dreaptă, unică, a lui $* a = a$. În pași de mai sus care implică reduceri, am indicat simbolurile ce se scot de pe stivă. De asemenea, după ce s-a efectuat o reducere, prin care simbolurile de pe stivă scoase au fost înlocuite cu un neterminal, starea ce este introdusă imediat după acel neterminal este dată de funcția *goto* care are ca argumente starea ce apare pe stivă înaintea celui neterminal și neterminalul respectiv.

PARSER LR PENTRU GRAMATICI SLR(1)

Gramaticile de tip $SLR(1)$ sunt gramatici de tip $LR(1)$ de o formă simplificată. Fie $G = (N, \Sigma, S, P)$ o gramatică independentă de context și extensia sa

$$G' = (N \cup \{S'\}, \Sigma \cup \{\#\}, S', P \cup \{S' \rightarrow S\}).$$

Definiție. Spunem că $\gamma \in (N \cup \Sigma)^*$ este un prefix viabil pentru G dacă în G există derivarea $S \xRightarrow[d]{*} \alpha A w \xRightarrow[d]{} \alpha \beta w$, $\alpha, \beta \in (N \cup \Sigma)^*$, $w \in \Sigma^*$ și γ este prefix al lui $\alpha \beta$.

Definiție. Numim configurație $LR(0)$ pentru G' structura simplă $[A \rightarrow \alpha. \beta]$, unde $A \rightarrow \alpha \beta \in P$. O configurație de forma $[A \rightarrow \alpha.]$ se numește finală.

Definiție. Spunem că $[A \rightarrow \alpha. \beta]$ este validă pentru prefixul viabil γ dacă în G avem:

$$a) \exists S \xRightarrow[d]{*} \delta A w \xRightarrow[d]{} \delta \alpha \beta w, w \in \Sigma^*$$

$$b) \gamma = \delta \alpha$$

Determinarea mulțimilor canonice $LR(0)$ pentru gramatica G (G')

Construim trei funcții, *closure*, *goto*, *config* în felul următor:

closure(I) // I mulțime de configurații $LR(0)$

```
{
     $J \leftarrow I$ ;
    do{
        for( fiecare  $A \rightarrow \alpha.B\beta \in J$ )
            for( fiecare  $B \rightarrow \gamma \in P$ )
                if( $B \rightarrow.\gamma \notin J$ )
                    adauga  $B \rightarrow.\gamma$  la  $J$ 
    }while ( se adauga noi configuratii la  $J$  )
    return  $J$ ;
} // end_closure (I)
```

goto(I, X) // I mulțime de configurații $LR(0)$, $X \in N \cup \Sigma$

```
{
     $J \leftarrow \{A \rightarrow \alpha X.\beta \mid A \rightarrow \alpha.X\beta \in I\}$ 
    return(closure( $J$ ));
} // end_goto
```

config(G) // returneaza multimea configuratiilor canonice $LR(0)$, C_G

```
{
     $C_G \leftarrow \{closure(\{S' \rightarrow.S\})\}$  cu closure( $\{S' \rightarrow.S\}$ ) nemarcata
    while( exista  $I \in C_G$  nemarcata )
    {
        marcheaza  $I$ ;
        for( fiecare  $X \in N \cup \Sigma$ )
            if(goto( $I, X$ )  $\neq \emptyset$  && goto( $I, X$ )  $\notin C_G$  )
                adauga goto( $I, X$ ) la  $C_G$  ca stare nemarcata;
    } //end_while
    return  $C_G$ ;
} // end_config
```

Tabela $SLR(1)$ (*action, goto*) pentru gramatica G

INPUT: $G = (N, \Sigma, S, P)$ o gramatică independentă de context și G' extensia sa

OUTPUT: tabelele *action, goto* asociate lui G (G')

ALGORITMUL:

1. Calculează mulțimile canonice $LR(0)$, $C_G = \{I_0, I_1, \dots, I_n\}$, unde $I_0 = closure(\{S' \rightarrow \cdot S\})$
2. Calculează mulțimile $Follow(A)$, $\forall A \in N$. **Se va inițializa $Follow(S)$ cu $\{\#\}$.**
3. Pentru fiecare j , $0 \leq j \leq n$, j stare ce corespunde lui I_j
 - 3.1. Dacă $[A \rightarrow \alpha \cdot a\beta] \in I_j$ && $goto(I_j, a) = I_k$, $a \in \Sigma$, atunci
 $action[j, a] = shift\ k$
 - 3.2. Dacă $[A \rightarrow \alpha \cdot] \in I_j$, $A \neq S'$, atunci
 $action[j, a] = reduce\ A \rightarrow \alpha$, $\forall a \in Follow(A)$
 - 3.3. Dacă $[S' \rightarrow \cdot S] \in I_j$, atunci
 $action[j, \#] = accept$
 - 3.4. $\forall x \in \Sigma \cup \{\#\}$, $action[j, x] = error$, în rest
 - 3.5. Pentru fiecare $A \in N$
 - 3.5.1. Dacă $goto(I_j, A) = I_k$, atunci
 $goto[j, A] = k$
 - 3.5.2. Dacă $goto(I_j, A) = \emptyset$, atunci
 $goto[j, A] = error$

Definiție. Dacă tabela *action* construită ca mai sus nu are intrări multiple, spunem că G este de tip $SLR(1)$.

EXEMPLU DE GRAMATICĂ $SLR(1)$

Fie G_2 gramatica cu producțiile:

- 1: $E \rightarrow TR$
- 2: $R \rightarrow +TR$
- 3: $R \rightarrow *TR$
- 4: $R \rightarrow \lambda$
- 5: $T \rightarrow n$

6: $T \rightarrow (E)$

- 1) Extindem G la G' : introducem productia $E' \rightarrow E$
- 2) Calculăm mulțimile *Follow* cu algoritmul din cursul 4, inițializând $Follow(E)$ cu $\{\#\}$. Obținem:

$Follow(X)$	Pasul 1	Pasul 2
E	$\#,)$	
T	$+, *, \#$	$)$
R	$\#$	$)$

- 3) Calculăm mulțimile canonice $LR(0)$ pentru G_2 :

$$I_0 = \left[\begin{array}{l} E' \rightarrow .E \xrightarrow{goto(I_0, E)} I_1 \\ E \rightarrow .TR \xrightarrow{goto(I_0, T)} I_2 \\ T \rightarrow .n \xrightarrow{goto(I_0, n)} I_3 \\ T \rightarrow .(E) \xrightarrow{goto(I_0, () } I_4 \end{array} \right]$$

$$I_1 = [E' \rightarrow E.]$$

$$I_2 = \left[\begin{array}{l} E \rightarrow T.R \xrightarrow{goto(I_2, R)} I_5 \\ R \rightarrow .+TR \xrightarrow{goto(I_2, +)} I_6 \\ R \rightarrow .*TR \xrightarrow{goto(I_2, *)} I_7 \\ R \rightarrow . \end{array} \right]$$

$$I_3 = [T \rightarrow n.]$$

$$I_4 = \left[\begin{array}{l} T \rightarrow (.E) \xrightarrow{goto(I_4, E)} I_8 \\ E \rightarrow .TR \xrightarrow{goto(I_4, T)} I_2 \\ T \rightarrow .n \xrightarrow{goto(I_4, n)} I_3 \\ T \rightarrow .(E) \xrightarrow{goto(I_4, () } I_4 \end{array} \right]$$

$$I_5 = [E \rightarrow TR.]$$

$$I_6 = \left[\begin{array}{l} R \rightarrow +.TR \xrightarrow{goto(I_6, T)} I_9 \\ T \rightarrow .n \xrightarrow{goto(I_6, n)} I_3 \\ T \rightarrow .(E) \xrightarrow{goto(I_6, () } I_4 \end{array} \right]$$

$$I_7 = \left[\begin{array}{l} R \rightarrow *.TR \xrightarrow{goto(I_7, T)} I_{10} \\ T \rightarrow .n \xrightarrow{goto(I_7, n)} I_3 \\ T \rightarrow .(E) \xrightarrow{goto(I_7, () } I_4 \end{array} \right]$$

$$I_8 = [T \rightarrow (E.) \xrightarrow{goto(I_8,)} I_{11}]$$

$$I_9 = \left[\begin{array}{l} R \rightarrow +T.R \xrightarrow{goto(I_9, R)} I_{12} \\ R \rightarrow .+TR \xrightarrow{goto(I_9, +)} I_6 \\ R \rightarrow .*TR \xrightarrow{goto(I_9, *)} I_7 \\ R \rightarrow . \end{array} \right]$$

$$I_{10} = \left[\begin{array}{l} R \rightarrow *T.R \xrightarrow{goto(I_{10}, R)} I_{13} \\ R \rightarrow .+TR \xrightarrow{goto(I_{10}, +)} I_6 \\ R \rightarrow .*TR \xrightarrow{goto(I_{10}, *)} I_7 \\ R \rightarrow . \end{array} \right]$$

$$I_{13} = [R \rightarrow^* TR.]$$

$$I_{11} = [T \rightarrow (E).]$$

$$I_{12} = [R \rightarrow +TR.]$$

Tabela de analiză sintactică $SLR(1)$ pentru G_2 :

	<i>action</i>						<i>goto</i>		
	+	*	n	()	#	E	T	R
0	Error	Error	shift 3	shift 4	error	error	1	2	Error
1	Error	Error	error	error	error	accept	error	error	Error
2	shift 6	shift 7	error	error	reduce 4	reduce 4	error	error	5
3	reduce 5	reduce 5	error	error	reduce 5	reduce 5	error	error	Error
4	Error	Error	shift 3	shift 4	error	error	8	2	Error
5	Error	Error	error	error	reduce 1	reduce 1	error	error	Error
6	Error	Error	shift 3	shift 4	error	error	error	9	Error
7	Error	Error	shift 3	shift 4	error	error	error	10	Error
8	Error	Error	error	error	shift 11	error	error	error	Error
9	shift 6	shift 7	error	error	reduce 4	reduce 4	error	error	12
10	shift 6	shift 7	error	error	reduce 4	reduce 4	error	error	13
11	reduce 6	reduce 6	error	error	reduce 6	reduce 6	error	error	error
12	Error	Error	error	error	reduce 2	reduce 2	error	error	error
13	Error	Error	error	error	reduce 3	reduce 3	error	error	error

Tabela *action* nu are intrări multiple, rezultă că G_2 este $SLR(1)$.

Analiza sintactică decurge exact ca pentru gramaticile $LR(1)$.