TEHNICI DE COMPILARE – CURSUL 6

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 (a se vedea cursul 3).
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.

GRAMATICI ȘI LIMBAJE DE TIP *LR*

Gramaticile de tip $LR(k)$ sunt gramatici neambigue, pentru care se pot implementa algoritmi de parsare <u>liniari</u> de tip bottom-up. Denumirea acestora provine din *Parsing from* **L***eft to Right using* **R***ightmost 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 <u>reducerii</u> 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:
$S \underset{d}{\overset{*}{\Rightarrow}} \alpha A u \underset{d}{\Rightarrow} \alpha\beta u = \gamma u, u \in \Sigma^*$ și $S \underset{d}{\overset{*}{\Rightarrow}} \alpha' A' u' \underset{d}{\Rightarrow} \alpha'\beta' u' = \alpha\beta v = \gamma v, v \in \Sigma^*$
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:

$$S \underset{d}{\overset{*}{\Rightarrow}} \alpha A u \underset{d}{\Rightarrow} \alpha\beta u \underset{d}{\overset{*}{\Rightarrow}} z \quad \text{și} \quad S \underset{d}{\overset{*}{\Rightarrow}} \alpha' A' u' \underset{d}{\Rightarrow} \alpha'\beta' u' = \alpha\beta u \underset{d}{\overset{*}{\Rightarrow}} z$$

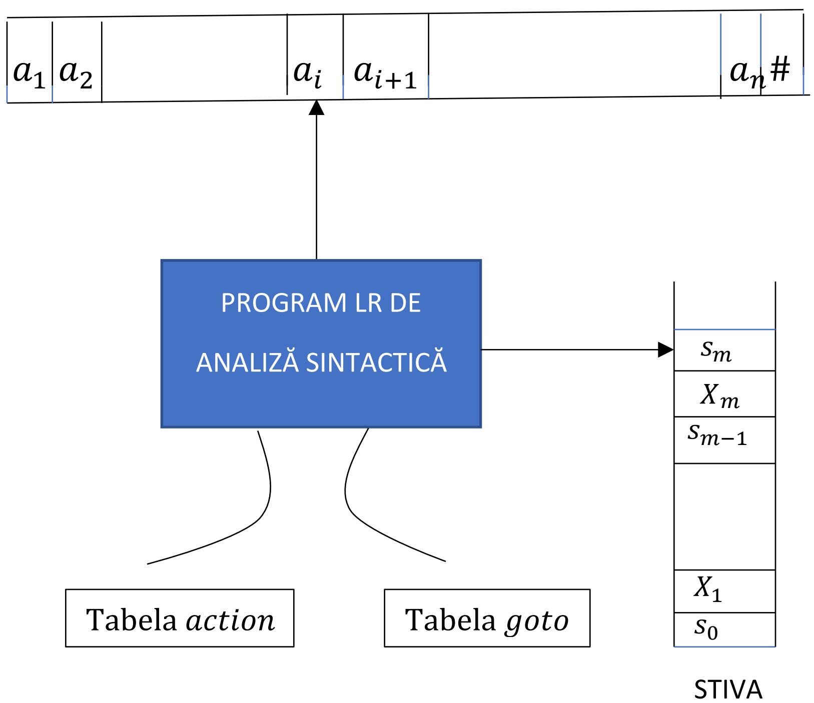
unde $\alpha\beta u \underset{d}{\overset{*}{\Rightarrow}} z$ este porțiunea ultimă comună a celor 2 derivări, $A \to \beta \neq A' \to \beta'$ sunt două producții distincte. Dar, deoarece G este $LR(k)$ și evident că $First_k(u) = First_k(u)$, rezultă că $\alpha = \alpha', A = A', \beta = \beta'$ adică $A \to \beta = A' \to \beta'$, 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 eficace
- un analizor sintactic de tip LR poate detecta rapid o eroare de sintaxă, permițând revenirea din eroare
- determinarea tabelei 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 definita prin $G' = (N \cup \{S'\}, \Sigma \cup \{\#\}, S', P \cup \{S' \to S\})$ cu $L(G) = L(G')$.
Algoritmul sintactic de tip LR pentru Gse 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 tabelei 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 $s_0X_1s_1 \ldots X_ms_m$, unde $X_i \in N \cup \Sigma, s_0, s_1, \ldots, s_m$ 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), be baza tabelelor $action$ 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

  $$(s_0 X_1 s_1 \ldots, X_m s_m, a_i a_{i+1} \ldots a_n \#, \pi)$$

  unde $X_1 \ldots X_m a_i \ldots a_n$ este o fomă sentențială a lui G dacă $a_1 \ldots a_n \in L(G)$ și π este o derivare dreaptă parțială a sa.
- Configurația inițială este $(s_0, a_1 \ldots a_n \#, \lambda)$, $a_1 \ldots a_n \in \Sigma^*$șirul analizat, s_0 starea inițială a AFD care recunoaște mulțimea prefixelor viabile.

Mișcările parserului:

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

Algoritmul, în pseudocod, este:

INPUT. Gramatica independentă de context $G = (N, \Sigma, S, P)$; G' extensia lui G, tabelele $action, goto$; w șirul analizat, $|w| = n, w = w_1 \ldots w_n, w_{n+1} = \#$.
Configurația inițială este: $(s_0, w\#, \lambda)$

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

$Q \leftarrow \emptyset$ // Q coadă
$i \leftarrow 1$ // 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
    }

$else \quad if(action[s,a] == reduce\ k{:}\ A \to \beta)\{$

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

$s' \leftarrow$ simbolul din varful stivei;

$push\ A;$

$push\ goto[s', A];$

$k \Rightarrow Q;$

}

$else \quad if(action[s,a] == "accept")$

$accept()$ // 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 <u>prefix viabil</u> pentru G dacă în G există derivarea $S \underset{d}{\overset{*}{\Rightarrow}} \alpha A w \underset{d}{\Rightarrow} \alpha\beta w, \alpha, \beta \in (N \cup \Sigma)^*, w \in \Sigma^*$ și γ este prefix al lui $\alpha\beta$.

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

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

a) $\exists S \underset{d}{\overset{*}{\Rightarrow}} \delta A w \underset{d}{\Rightarrow} \delta\alpha\beta w, w \in \Sigma^*$
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 ← I;
        do{
                for( fiecare A → α.Bβ; a ∈ J)
                        for( fiecare B → γ ∈ P)
                                for( fiecare b ∈ First(βa)
                                        if(B →.γ; b ∉ J)
                                                adauga B →.γ; b la J
                }while ( se adauga noi configuratii la J )
        return J;
} // end_closure (I)
```

- $goto(I, X)$ // I mulțime de configurații $LR(1), X \in N \cup \Sigma$

```
{
        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 $I_0 = closure(\{S' \to .S; \#\})$
2. Pentru fiecare $j, 0 \le j \le n, j$ stare ce corespunde lui I_j
   - 2.1. Daca $[A \to \alpha.a\beta; b] \in I_j$ && $goto(I_j, a) = I_k, a \in \Sigma$, atunci
     $action[j, a] = shift\ k$
   - 2.2. Daca $[A \to \alpha.; a] \in I_j, A \neq S'$, atunci
     $action[j, a] = reduce\ A \to \alpha$
   - 2.3. Daca $[S' \to S.; \#] \in I_j$,
     $action[j, \#] = accept$
   - 2.4. $\forall x \in \Sigma \cup \{\#\}$, $action[j, x] = error$, in rest
   - 2.5. Pentru fiecare $A \in N$
     - 2.5.1. Daca $goto(I_j, A) = I_k$, atunci
       $goto[j, A] = k$
     - 2.5.2. Daca $goto(I_j, A) = \emptyset$, atunci
       $goto[j, A] = error$

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

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

Definiție. Spunem că $\gamma \in (N \cup \Sigma)^*$ este un <u>prefix viabil</u> pentru G dacă în G există derivarea $S \overset{*}{\underset{d}{\Rightarrow}} \alpha A w \underset{d}{\Rightarrow} \alpha\beta w, \alpha, \beta \in (N \cup \Sigma)^*, w \in \Sigma^*$ și γ este prefix al lui $\alpha\beta$.

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

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

a) $\exists S \underset{d}{\overset{*}{\Rightarrow}} \delta A w \underset{d}{\Rightarrow} \delta\alpha\beta w, w \in \Sigma^*$

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 $LR(0)$
  {
  $\quad J \leftarrow I;$
  $\quad do\{$
  $\quad\quad for($ fiecare $A \to \alpha.B\beta \in J)$
  $\quad\quad\quad for($ fiecare $B \to \gamma \in P)$
  $\quad\quad\quad\quad if(B \to .\gamma \notin J)$
  $\quad\quad\quad\quad\quad$ adauga $B \to .\gamma$ la J
  $\quad\}while$ (se adauga noi configuratii la J)
  $\quad return\, J;$
  } // end_closure (I)

- $goto(I, X)$ // I mulțime de configurații $LR(0), X \in N \cup \Sigma$
  {
  $\quad J \leftarrow \{A \to \alpha X.\beta \mid A \to \alpha.X\beta \in I\}$
  $\quad return\big(closure(J)\big);$
  } // end_goto

- $config(G\ )$ // returneaza multimea configuratiilor canonice $LR(0)$, C_G
  {
  $\quad C_G \leftarrow \{closure(\{S' \to .S\})\}$ cu $closure(\{S' \to .S\})$ nemarcata
  $\quad while($ exista $I \in C_G$ nemarcata)
  $\quad\quad$ {

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 $\boldsymbol{SLR(1)(action, goto)}$ pentru gramatica $\boldsymbol{G}$

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 mulțimile canonice $LR(0)$, $C_G = \{I_0, I_1, \dots, I_n\}$, unde $I_0 = closure(\{S' \to .S\})$
2. Calculează mulțimile $Follow(A)$, $\forall A \in N$. **Se va inițializa $\boldsymbol{Follow(S)}$ cu {#}.**
3. Pentru fiecare $j, 0 \leq j \leq n, j$ stare ce corespunde lui I_j

   3.1. Daca $[A \to \alpha.a\beta] \in I_j$ && $goto(I_j, a) = I_k$, atunci

   $action[j, a] = shift\ k$

   3.2. Daca $[A \to \alpha.] \in I_j,\ A \neq S'$, atunci

   $action[j, a] = reduce\ A \to \alpha,\ \forall a \in Follow(A)$

   3.3. Daca $[S' \to S.] \in I_j$, 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. Daca $goto(I_j, A) = I_k$, atunci

   $goto[j, A] = k$

   3.5.2. Daca $goto(I_j, A) = \emptyset$, 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 $LR(1)$

Fie G_1 gramatica cu producțiile:

1: $S \to \quad L = R$
2: $S \to \quad R$
3: $L \to \quad * R$
4: $L \to \quad a$
5: $R \to \quad L$

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

$$I_0 \quad \begin{bmatrix} S' \to & .S;\# \\ S \to & .L = R;\# \\ S \to & .R;\# \\ L \to & .*R;= \\ L \to & .a;= \\ R \to & .L;\# \\ L \to & .*R;\# \\ L \to & .a;\# \end{bmatrix} = \begin{bmatrix} S' \to & .S;\# & \xrightarrow{goto(I_0,S)} I_1 \\ S \to & .L = R;\# & \xrightarrow{goto(I_0,L)} I_2 \\ S \to & .R;\# & \xrightarrow{goto(I_0,R)} I_3 \\ L \to & .*R;=|\# & \xrightarrow{goto(I_0,*)} I_4 \\ L \to & .a;=|\# & \xrightarrow{goto(I_0,a)} I_5 \\ R \to & .L;\# & \xrightarrow{goto(I_0,L)} I_2 \end{bmatrix}$$

$$I_1 = [S' \to \quad S.;\#]$$

$$I_2 = \begin{bmatrix} S \to & L. = R;\# \to I_6 \\ R \to & L.;\# \end{bmatrix}$$

$$I_3 = [S \to \quad R.;\#]$$

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

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

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

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

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

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

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

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

$I_{12} = [L \rightarrow \quad a.; \#]$ $\qquad$ $I_{13} = [L \rightarrow \quad * R.; \#]$

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

	$action$				$goto$		
	=	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 "$* a = a$":

$$(0, * a = a\#, \lambda) \overset{s_4}{\vdash} (0 * 4, a = a\#, \lambda) \overset{s_5}{\vdash} (0 * 4\, \underbrace{a5}, = a\#, \lambda) \overset{r_4}{\vdash} (0 * 4\, \underbrace{L8}, = a\#, 4)$$

$$\overset{r_5}{\vdash} (0\, \underbrace{* 4R7}, = a\#, 54) \overset{r_3}{\vdash} (0L2, = a\#, 354) \overset{s_6}{\vdash} (0L2 = 6, a\#, 354)$$

$$\overset{s_{12}}{\vdash} (0L2 = 6\, \underbrace{a12}, \#, 354) \overset{r_4}{\vdash} (0L2 = 6\, \underbrace{L\ 10}, \#, 4354) \overset{r_5}{\vdash} (0\, \underbrace{L2 = 6R9}, \#, 54354)$$

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

Șirul 154354 reprezintă derivarea dreaptă, unică, a lui $* a = a$. În pașii de mai sus care implică reduceri, am indicat simbolurile ce se scot de pe stivă. De asemenea, dupa 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 acelui neterminal și neterminalul respectiv.