LABORATOR NR 5

GRAMATICI LR(k). ANALIZA SINTACTICĂ LR(k).

Gramaticile *LR* au fost introduse în 1965 de către Knuth iar clasa limbajelor generate de ele este clasa limbajelor independente de context deterministe. Analiza *LR* prezintă avantajul generalității, toate limbajele de programare ce acceptă o definiție sintactică *BNF* fiind analizabile *LR*.

Pentru efectuarea unei analize sintactice de tip *LR* se dispune de următoarele informații:

1. o poziție inițială în cuvântul de analizat pe care o notăm cu *p*,
2. întregul context stânga al cuvântului sursă, adică $a_1 a_2 ... a_p$
3. următoarele *k* simboluri ale sursei situate după poziția *p*, adică $a_{p+1} a_{p+2} ... a_{p+k}$,
4. analiza ce se efectuează este o analiză la dreapta; adică la fiecare pas, neterminalul care derivează este cel mai din dreapta.

Gramatica G trebuie să aibă astfel de proprietăți încât informațiile 1-4 să ne asigure următoarele:

- dacă p indică sau nu limita dreaptă a părții reductibile,
- dacă p indică limita dreaptă a părții reductibile atunci 1-4 determină şi limita din stânga,
- dacă a fost determinată partea reductibilă 1-4 determină şi producția ce va fi utilizată pentru reducere.

Fie $G = (N, \Sigma, P, S)$ o gramatică independentă de context şi $w \in L(G)$. Pentru $A \in N$ vrem să determinăm şirul $\alpha_0, \alpha_1, ..., \alpha_m$ astfel încât: $S = \alpha_0 \overset{d}{\Rightarrow} \alpha_1 \overset{d}{\Rightarrow} ... \overset{d}{\Rightarrow} \alpha_m = w$. Fie $\alpha_{i-1} = \alpha A x$ si $\alpha_i = \alpha \beta x$; informațiile 1-4 trebuie să determine în mod unic producția $A \to \beta$ folosită în derivarea $\alpha_{i-1} \overset{*}{\Rightarrow} \alpha_i$. În cazul gramaticilor LR toate derivările sunt la dreapta.

Definiție. Fie $G = (N, \Sigma, P, S)$ o gramatică independentă de context. Se numeşte gramatică extinsă a lui G, gramatica $G' = (N \cup \{S'\}, \Sigma, P \cup \{S' \to S\}, S')$ unde $S' \notin N$. Producția $S' \to S$ o vom numerota cu *0* iar celelalte cu *1,2,...,p*.

Definiție. Fie $G = (N, \Sigma, P, S)$ o gramatică independentă de context şi $G' = (N', \Sigma, P', S')$ gramatica sa extinsă.*G* este *LR(k)*, $k \geq 0$, dacă:

(1) $S' \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha \beta w$

(2) $S' \overset{*}{\Rightarrow} \gamma B x \Rightarrow \gamma \delta x = \alpha \beta y$

(3) $PRIM_k(w) = PRIM_k(y)$

implică $\alpha A y = \gamma B x$ (adică $\alpha = \gamma, A = B, x = y$).

Definiție. Cuvântul $\gamma \in (N \cup \Sigma)^*$ este un prefix viabil în gramatica $G = (N, \Sigma, P, S)$ dacă $S \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha \beta w$ şi γ este un prefix al lui $\alpha\beta; \alpha\beta$ se numeşte partea deschisă.

Definiție. Fie $G = (N, \Sigma, P, S)$ o gramatică independentă de context. Vom spune că $[A \to \beta_1 . \beta_2, u]$ este linie *LR(k)*, dacă $(A \to \beta_1 \beta_2) \in P$ şi $u \in \Sigma^{*k}$. $A \to \beta_1 . \beta_2$ se numeşte nucleul iar *u* şirul de anticipare al liniei.

Definiție. Linia *LR(k)* $[A \to \beta_1 . \beta_2, u]$ este validă pentru prefixul viabil $\alpha\beta_1$ dacă există o derivare de forma $S' \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha \beta_1 \beta_2 w$ şi $u = PRIM_k(w)$.

Definiție. Funcția $EFF : (N \cup \Sigma)^* \to P_f(\Sigma^*)$ (ε-free first) este o restricție a funcției PRIM si este definită astfel:

$EFF_k(\alpha) = PRIM_k(\alpha)$ *daca* $\alpha \notin N(N \cup \Sigma)^*$ şi

$EFF_k(\alpha) = \{w \mid \alpha \overset{*}{\Rightarrow} wx,\ w = PRIM_k(wx)$ şi ultima producție folosită nu este ε-producție când $\alpha \in N(N \cup \Sigma)^*\}$.

Lemă. Fie $G = (N, \Sigma, P, S')$ o gramatică extinsă care nu este *LR(k).* Atunci:

(1) $S' \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha \beta w$

(2) $S' \overset{*}{\Rightarrow} \gamma B x \Rightarrow \gamma \delta x = \alpha \beta y$

(3) $PRIM_k(w) = PRIM_k(y)$

(4) $|\alpha\beta| \le |\gamma\delta|$

implică $\alpha Ay \neq \gamma Bx$.

Teoremă. O gramatică $G = (N, \Sigma, P, S)$ este *LR(k)* dacă şi numai dacă este indeplinită următoarea condiție, pentru orice $u \in \Sigma^{*k}$: dacă $[A \to \beta., u]$ este linie LR(k) validă pentru prefixul viabil $\alpha\beta$ atunci nu există nicio altă linie $[A \to \beta_1.\beta_2, v]$ validă pentru $\alpha\beta$ cu $u \in EFF_k(\beta_2 v)$.

Analiza sintactică de tip LR

Gramaticile *LR* au fost introduse în 1965 de către Knuth iar clasa limbajelor generate de ele este clasa limbajelor independente de context deterministe. Analiza *LR* prezintă avantajul generalității, toate limbajele de programare ce acceptă o definiție sintactică *BNF* fiind analizabile *LR*.

Efectuarea unei analize de tip *LR* presupune următoarele informații:o poziție inițială în cuvântul de analizat pe care o notăm cu *p*, contextul stânga al cuvântului sursă, adică $a_1 a_2 ... a_p$ si următoarele *k* simboluri ale sursei situate după poziția *p*, adică $a_{p+1} a_{p+2} ... a_{p+k}$. Analiza ce se efectuează este o analiză la dreapta. Aceste proprietăți ne asigura: dacă *p* indică sau nu limita dreaptă a părții reductibile, determină şi limita din stânga, iar dacă a fost determinată partea reductibilă, determină şi producția ce va fi utilizată pentru reducere.

Fie $G = (N, \Sigma, P, S)$ şi $w \in L(G)$. Pentru $A \in N$ se determină şirul $\alpha_0, \alpha_1, ..., \alpha_m$ astfel încât: $S = \alpha_0 \overset{d}{\Rightarrow} \alpha_1 \overset{d}{\Rightarrow} .... \overset{d}{\Rightarrow} \alpha_m = w$. Fie $\alpha_{i-1} = \alpha Ax$ şi $\alpha_i = \alpha\beta x$; informațiile trebuie să determine în mod unic producția $A \to \beta$ folosită în derivarea $\alpha_{i-1} \overset{d}{\Rightarrow} \alpha_i$. În cazul gramaticilor *LR* toate derivările sunt la dreapta.

Definiție. Fie $G=(N,\Sigma,P,S)$ o GIC. Se numeşte gramatică extinsă a lui G,

gramatica $G'=(N\cup\{S'\},\Sigma,P\cup\{S'\to S\},S')$ unde $S'\notin N$. Producția $S'\to S$ se numeroteza cu 0 iar celelalte cu 1, 2,…, *p*.

Definiție. Fie $G=(N,\Sigma,P,S)$ o GIC $G'=(N',\Sigma,P',S')$ gramatica sa extinsă. G este LR(k), $k\geq 0$, dacă:

1) $S'\overset{*}{\Rightarrow}\alpha Aw\Rightarrow\alpha\beta w$

2) $S'\overset{*}{\Rightarrow}\gamma Bw\Rightarrow\gamma\delta x\Rightarrow\alpha\beta y$

3) $PRIM_k(w)=PRIM_k(y)$

implică $\alpha Ay=\gamma Bx$ (adică $\alpha=\gamma, A=B$ *şi* $x=y$).

Definiție. Cuvântul $\gamma\in(N\cup\Sigma)^*$ este un *prefix viabil* în gramatica $G=(N,\Sigma,P,S)$ dacă $S\overset{*}{\Rightarrow}\alpha Aw\Rightarrow\alpha\beta w$ şi γ este un prefix al lui $\alpha\beta;\alpha\beta$ se numeşte *parte deschisă*.

Definiție. Fie $G=(N,\Sigma,P,S)$ o GIC. Vom spune că $[A\to\beta_1.\beta_2,u]$ este linie LR(k), dacă $(A\to\beta_1\beta_2)\in P$ *şi* $u\in\Sigma^{*k}$. $A\to\beta_1.\beta_2$ se numeşte *nucleul* iar *u şirul de anticipare* al liniei.

Definiție. Linia LR(k) $[A\to\beta_1.\beta_2,u]$ este validă pentru prefixul viabil $\alpha\beta_1$ dacă există o derivare de forma $S\overset{*}{\Rightarrow}\alpha Aw\Rightarrow\alpha\beta_1\beta_2 w$ şi $u=PRIM_k(w)$.

Definiție. Funcția $EFF:(N\cup\Sigma)^*\to P_f(\Sigma^*)$ (ε-free first) este o restricție a funcției PRIM şi este definită astfel:

$EFF_k(\alpha) = PRIM_k(\alpha)$ *dacă* $\alpha \notin N(N \cup \Sigma)^*$ şi

$EFF_k(\alpha) = \left\{ w / \alpha \overset{*}{\Rightarrow} wx, w = PRIM_k(wx) \right\}$ şi ultima producţie folosită nu este ε - producţie când $\alpha \in N(N \cup \Sigma)^*$

Lemă. Fie $G = (N, \Sigma, P, S')$ o gramatică extinsă care nu este LR(k). Atunci:

1) $S' \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta w$

2) $S' \overset{*}{\Rightarrow} \gamma B w \Rightarrow \gamma\delta x \Rightarrow \alpha\beta y$

3) $PRIM_k(w) = PRIM_k(y)$

4) $|\alpha\beta| \le |\gamma\delta|$

implică $\alpha A y \neq \gamma B x$.

Teoremă. *O gramatică* $G = (N, \Sigma, P, S)$ *este LR(k) dacă şi numai dacă este îndeplinită următoarea condiţie,pentru orice* $u \in \Sigma^{*k}$ *: dacă* $[A \to \beta., u]$ *este linia LR(k)*
validă pentru prefixul viabil $\alpha\beta$ *atunci nu există nici-o altă linie* $[A \to \beta_1.\beta_2, v]$ *validă pentru* $\alpha\beta$ *cu* $u \in EFF_k(\beta_2 v)$.

Testarea condiţiei LR(k)

Fie G o GIC şi γ un prefix viabil. Se noteaza cu $V_k^G(\gamma)$ mulţimea liniilor LR(k) valide pentru prefixul viabil γ .

Algoritm. -calculează mulţimile $V_k^G(\gamma)$

Intrare : $G = (N, \Sigma, P, S)$ o GIC, $\gamma = x_1 x_2 ... x_n \in (N \cup \Sigma)^*$ şi $n \ge 0$

Ieşire : $V_k^G(\gamma)$

Metoda : Se calculează succesiv

$V_k^G(\varepsilon), V_k^G(x_1), V_k^G(x_1 x_2), ..., V_k^G(x_1 x_2 ... x_n)$

Pasul 1. Se construieşte mulţimea $V_k^G(\varepsilon)$

(a) Dacă $(S \to \alpha) \in P$, se include în $V_k^G(\varepsilon)$ linia $[S \to .\alpha, \varepsilon]$

(b) Dacă $[A \to .B\beta_2, u] \in V_k^G(\varepsilon)$ şi $(B \to \beta) \in P$, atunci pentru $\forall x \in PRIM_k(\beta_2 u)$ se include în $V_k^G(\varepsilon)$ linia $[B \to .\beta, x]$, dacă nu a fost deja inclusă.

(c) Se repetă pasul 1.b până când nici-o linie nu mai poate fi adăugată la $V_k^G(\varepsilon)$.

Pasul 2. Se presupune că a fost calculată mulţimea $V_k^G(x_1 x_2 ... x_{i-1})$ şi se calculeaza $V_k^G(x_1 x_2 ... x_i)$ astfel:

(a) Dacă $[A \to \beta_1 . x_i \beta_2, v] \in V_k^G(x_1 x_2 ... x_{i-1})$ atunci se include în $V_k^G(x_1 x_2 ... x_i)$ linia $[A \to \beta_1 x_i . \beta_2, v]$.

(b) Dacă $[A \to \beta_1 . B\beta_2, u] \in V_k^G(x_1 ... x_i)$ şi $(B \to \beta) \in P$ atunci se include în $V_k^G(x_1 x_2 ... x_i)$ linia $[B \to .\beta, x]$ pentru $\forall x \in PRIM_k(\beta_2 u)$ dacă aceasta nu a fost deja inclusă.

(c) Se repetă pasul *(2.b)* până când nu se mai poate adăuga nici-o linie la mulţimea $V_k^G(x_1 x_2 ... x_i)$.

Definiţie. Fie gramatica $G = (N, \Sigma, P, S)$ şi $k \geq 0$ $\mathsf{A} = V_k^G(\gamma)$, unde $\gamma \in (N \cup \Sigma)^*$. Se defineste funcţia GOTO astfel: GOTO(A,x)=A′, unde $\mathsf{A}' = V_k^G(\gamma x)$ şi $x \in N \cup \Sigma$.

Algoritm. - metodă de calcul a mulţimii de linii LR(k).

Intrare : GIC $G = (N, \Sigma, P, S)$ şi $k \geq 0$, număr de intrare

Ieşire : $\mathsf{S} = \{\mathsf{A} \mid \mathsf{A} \in V_k^G$ şi γ este prefix viabil al lui G$\}$

Metoda : Iniţial $\mathsf{S} = \varnothing$

Pasul 1. Se include în S mulțimea $V_k^G(\varepsilon)$ nemarcată.

Pasul 2. Dacă $A \in S$ este nemarcată, atunci se marchează A după ce se calculează pentru fiecare $\alpha \in N \cup \Sigma$ mulțimea A= GOTO(A,x). Dacă $A \neq \emptyset$ şi nu există deja în S se adaugă A la S, nemarcată.

Pasul 3. Se repetă pasul (2) până când toate elementele lui S sunt marcate.

Mulțimea S se numeşte *colecția canonică* de mulțimi de linii LR(k), pentru gramatica G.

Definiție. Fie $G = (N, \Sigma, P, S)$ o GIC şi *k* un număr întreg nenegativ. O mulțime A de linii LR(k) se numeşte *consistentă* dacă nu conține două linii de forma $[A \to \beta., u]$ şi $[B \to \beta_1.\beta_2, v]$ cu $u \in EFF_k(\beta_2 v)$.

Algoritm –testează condiția LR(k)

Intrare : $G = (N, \Sigma, P, S)$ o GIC şi $k \geq 0$ un număr întreg

Ieşire : "DA" dacă G este LR(k) şi "NU" în caz contrar

Metoda :

Pasul 1. Se calculează colecția canonică de mulțimi de linii LR(k).

Pasul 2. Se examinează fiecare mulțime de linii LR(k) din S şi se determină dacă este consistentă.

Pasul 3. Dacă toate mulțimile din S.. Sunt consistente se răspunde "DA"; în caz contrar se răspunde "NU".

Definiție. Fie $G = (N, \Sigma, P, S)$ o GIC şi S colecția canonică de mulțimi de linii LR(k).T(A), *tabelul LR(k)* asociat mulțimii $A \in S$, este o pereche de funcții $< f, g >$ unde *f* este funcția de trecere iar *g* este funcția GOTO:

(1) $f:\Sigma^{*k} \to \{eroare, trecere, acceptare\} \cup \{i / i \text{ este o productie din } P, i \geq 1\}$

unde

(a) $f(u) = trecere$ dacă $[A \to \beta_1.\beta_2, v] \in$ A $, \beta_2 \notin \varepsilon$ şi $u \in EFF_k(\beta_2 v)$.

(b) $f(u) = i$ dacă $[A \to \beta, u] \in$ A şi $A \to B$ este producția cu numărul i .

(c) $f(\varepsilon) = acceptare$ dacă $[S' \to S., \varepsilon] \in$ A .

(d) $f(u) = eroare$ în alte cazuri.

(2) funcția *g* determină următorul tabel ce va fi folosit în analiză; ea pune în corespondență unui element din $N \cup \Sigma$ un tabel LR(k) sau eroare:

$g(x) = GOTO$ (A,x) dacă GOTO(A,x)≠∅.

$g(x) = eroare$ dacă GOTO(A,x)≠∅.

Algoritm. –efectuează analiza sintactică LR.

Intrare : Mulțimea T de tabele LR(k) corespunzătoare gramaticii $G = (N, \Sigma, P, S)$ cu tabelul inițial T_0=T(A_0), $A_0 = V_k^G(\varepsilon)$ şi cuvântul inițial $w \in \Sigma^*$.

Ieşire : Analiza sintactică la dreapta dacă $w \in L(G)$ şi "eroare" în caz contrar.

Metoda : Algoritmul lucrează cu o bandă de intrare, o bandă de ieşire şi una pushdown. Configurația inițială este (T_0, w, ε). Se execută pasii (1) şi (2) până când se obține un răspuns de acceptare sau de eroare.

Pasul 1. Se determină $u = PRIM_k$ (porțiunea din banda de intrare rămasă de citit).

Pasul 2. Fie $T_i =< f, g >$ linia înscrisă în vârful benzii pushdown;

(a) Dacă $f(u) = trecere$, atunci primul simbol disponibil *x* de pe banda de intrare se trece în vârful benzii pushdown. Calculează $g(x) = T_j$ şi înscrie T_j în vârful benzii pushdown şi apoi treci la pasul (1).

(b) Dacă $f(u)=i$ şi $A\to\alpha$ este regula *i* din P, atunci şterge $2|\alpha|$ simboluri de pe banda pushdown şi înscrie *i* pe banda de ieşire. Fie $T_j =< f_j, g_j >$ tabelul rămas în vârful benzii pushdown; determină $g_j(A)=T'$, înscrie pe banda pushdown pe AT' şi mergi la pasul (1).

Dacă $g_j(A)=eroare$, opreşte algoritmul şi cheamă, eventual, o rutină de tratare a erorii.

(c) Dacă $f(u)=eroare$, opreşte algoritmul şi cheamă, eventual, o rutină de tratare a erorii.

(e) Dacă $f(u)=acceptare$, opreşte algoritmul şi emite $\tilde{\pi}$, unde π este secvența de pe banda de ieşire.

Gramatici LR(1)

a) Gramatici LR(1)

Liniile LR(0) sunt de forma $[A\to\beta_1.\beta_2]$ unde $A\to\beta_1\beta_2$ este regula de producție; deci o linie LR(0) conține doar nucleul inei linii LR(k), $k\geq 1$.

Linia $[A\to\beta_1.\beta_2]$ este validă pentru prefixul viabil $\alpha\beta_1$ dacă există derivațiile $D\overset{*}{\Rightarrow}\alpha Aw\Rightarrow\alpha\beta_1\beta_2 w$.

Algoritm . -calculează $V_0(\gamma)$

Pasul 1. Construim mulțimea $V_0(\varepsilon)$

(a) Dacă $(S\to\alpha)\in P$ atunci $[S\to.\alpha]\in V_0(\varepsilon)$.

(b) Dacă $[A\to.B\beta_2]\in V_0(\varepsilon)$ şi $(B\to\beta)\in P$ atunci include în $V_0(\varepsilon)$ şi linia $[B\to.\beta]$.

(c) Repetă pasul (1.b) până când nu se mai adaugă nici o linie la $V_0(\varepsilon)$

Pasul 2. Calculează $V_0(x_1....x_i)$ după ce a fost calculată mulțimea $V_0(x_1....x_{i-1})$.

(a) Dacă $[A \to \beta_1 . x_i \beta_2] \in V_0(x_1 ... x_{i-1})$, atunci include în $V_0(x_1 .... x_i)$ linia $[A \to \beta_1 x_i . \beta_2]$.

(b) Dacă $[A \to \beta_1 . B\beta_2] \in V_0(x_1 ... x_i)$ şi $(B \to \beta) \in P$, atunci include în $V_0(x_1 .... x_i)$ linia $[B \to . \beta]$.

(c) Repetă pasul (2.b) până când nu se mai adaugă nici-o linie la $V_0(x_1 .... x_i)$.

Gramaticile LR(0) nu sunt suficient de puternice pentru a putea construi analizatori sintactici care să funcţioneze în mod determinist. Liniile LR(0) pot fi punctul de pornire pentru obţinerea analizei LR care priveşte un simbol spre dreapta.

b) **Gramatici** SLR(1)

Fiind cunoscută colecţia canonică de linii LR(0), dăm în continuare algoritmi pentru construirea tabelului şi analizei sintactice SLR(1). Tabelul de analiză SLR(1) presupune construirea funcţiilor *f* şi *g*. Pentru uşurinţa exprimării vom considera că cele două funcţii sunt de două variabile, prima variabilă reprezentând indicele mulţimii din colecţia canonică.

Algoritm. -construieşte funcţiile SLR(1)

Intrare : Colecţia canonică de linii LR(0) $S = (A_0, \ldots, A_n)$

Ieşire : Tabelul de analiză sintactică SLR(1).

Metoda :

Pasul 1. Se execută pasii (2) şi (3) pentru fiecare $A_i \in S$

Pasul 2. Determinarea funcţiei *f*

(a) Dacă $[A \to \alpha . x \beta] \in A_i$, unde $x \in \Sigma \cup \{\varepsilon\}$ şi $GOTO(A_i, x) = A_j$ atunci $f(i, x) = trecere$.

(b) Dacă $[A \to \alpha .] \in A_i$, unde $A \to \alpha$ este producţia cu numărul *j* atunci $f(i.x) = j$ (reducere cu regula *j*) pentru orice $x \in URM_1(A)$.

(c) Dacă $[S \to \alpha.] \in A_i$ atunci $f(i, \varepsilon) = acceptare$

(d) f ia valoarea *eroare* în alte cazuri.

Pasul 3. Determinarea funcției g

(a) $g(i, x) = j$ dacă GOTO(A_i,x)=A_j

(b) g are valoarea *eroare* în alte cazuri.

Definiție. Dacă prin aplicarea algoritmului de mai sus nu apar conflicte în construcția lui f, gramatica se numeşte SLR(1). Un conflict se întâlneşte când funcția f are valori multiple; conflicte posibile sunt reducere-trecere.

Deoarece trecerea de la un element la altul al colecției canonice poate fi exprimată (foarte sugestiv) printr-un automat finit, vom numi elementele colecției canonice *stări.*

Algoritmul de analiză sintactică foloseşte următoarele notații.

Bpd - banda pushdown pe care se depun indicii mulțimilor din colecția canonică

Urmcar - funcție ce returnează următorul caracter din colecția canonică

Curent - caracterul curent din cuvântul de intrare

Topv - funcția ce returnează starea aflată în vârful benzii pushdown

Push - procedura ce depune o stare în vârful benzii pushdown

Pop - procedura care şterge vârful benzii pushdown

Stânga - vector ce conține simbolurile din membrul stâng al producțiilor

Dreapta - vector ce conține simbolurile din partea dreaptă a fiecărei producții

T,I - variabile locale.

Algoritm –construieşte analiza sintactică SLR(1)

Intrare : Tabelul de analiză SLR(1)

Ieşire : Analiza sintactică SLR(1)

Metoda :

Pasul 1. Inițializarea benzii bpd

Push (bpd,0)

Pasul 2. Selectarea primului simbol din cuvântul de intrare

curent ← Urmcar

Pasul 3. Construirea arborelui de derivare, dacă este posibil repetă pasul (4) până când se ajunge la starea de acceptare (='a') sau de eroare.

Pasul 4. If *f* (Topv(bpd),Curent)='t' (trecere)

```
then
        begin
                T=g(Topv(bpd),Curent)
                Call Push (bpd,T)
                curent ← Urmcar
        end
else
  begin
  if f (Topv(bpd), curent)=k
        then
         begin
           write(k)
           for i=1,2,…, Dreapta (k)
                call Pop(bpd)
              T ← g(Topv(bpd), Stânga(k))
               call Push(bpd,T)
         end
       else
        begin
                if f (Topv(bpd), curent)='a'
```

```
        then
            write (‘cuvânt acceptat’)
        else
            write (‘eroare’)
    end
end.
```

c) Gramatici LALR(1)

Dându-se o gramatică extinsă şi colecția canonică de mulțimi de linii LR(1) se poate construi tabelul de analiză. Lipsa conflictelor din acest tabel este echivalentă cu faptul că gramatica este LR(1). Pentru aceeaşi gramatică, tabelul LR(1) este mai mre ca dimensiune decât tabelul SLR(1). Deoarece clasa gramaticilor SLR(1) este strict inclusă în clasa gramaticilor LR(1), dacă analiza SLR(1) eşuează se poate încerca analiza LR(1). Totuşi, se preferă analiza LALR(1), bazată pe gramatici LALR (în engleză lookhead LR grammars). Ea este mai avantajoasă deoarece tabelul LALR(1) este mai mic în comparație cu cel LR(1). De asemenea, spre deosebire de gramaticile SLR(1), gramaticile LALR(1) acoperă o clasă mai mare de limbaje ce include practic toate construcțiile sintactice folosite în limbajele de programare cunoscute.

Fie $A_i = \{[A \to \alpha., a]$ şi $A_j = \{[A \to \alpha., b]$ două elemente ale colecției canonice de linii LR(1). Liniile din cele două mulțimi au acelaşi nucleu, dar diferă prin şirul de anticipare (*a* şi recursiv *b*). După reducere, în funcție de elementul din vârful stivei şi simbolul de intrare, se ajunge în stări diferite.

Dacă se renunță la verificarea simbolului de intrare, cele două mulțimi de linii LR, A_i şi A_j sunt echivalente; putem să le înlocuim cu o nouă

mulțime $A_{i,j}=\{[A \to \alpha., a/b]\}$. Mai general, două mulțimi din colecția canonică LR(1) pot fuziona dacă au aceleaşi nuclee ale elementelor componente. Repetând această operație până când nu mai există stări cu nuclee identice, ajungem la un tabel SLR(1). Deoarece funcția GOTO depinde doar de nucleu, toate referirile la T_i şi T_j din definirea funcției *g* vor fi înlocuite prin $T_{i,j}$.

TEME PROPUSE

1. Implementați algoritmul V_k^G pentru calculul colecției canonice de linii LR(k).
2. Implementați algoritmul pentru testarea condiției LR k.
3. Implementați algoritmul pentru analiza sintactică LR k.
4. Implementați algoritmul de analiză sintactică SLR(1)