TEHNICI DE COMPILARE – CURSUL 4
METODA DE ANALIZĂ SINTACTICĂ TOP-DOWN

PARSER RECURSIV DESCENDENT GENERAL

Un parser recursiv descendent pentru o gramatică independentă de context $G=(N, \Sigma, S, P)$ constă din implementarea unei funcții pentru fiecare neterminal. Execuția începe cu apelul funcției corespunzătoare simbolului de start. Parserul nu funcționează pentru gramatici recursive la stânga. Fie $w = z\#$, unde $w \in \Sigma^*$ șirul analizat, iar # este un simbol nou (echivalentul lui EOF). tok este o variabila globală ce conține token-ul curent. Funcția getNextToken furnizează token-ul următor. Pentru un neterminal $A \in N$, funcția este:

```
void A() {
    while (true)
      {
        if ( mai exista productie A→ X_1 ⋯ X_k din P neanalizata)
           {
            for (i = 1, … , k) {
            if ( X_i este neterminal )
                    X_i() // se apeleaza functia corespunzatoare
           else if ( X_i == tok ) // daca simbolul curent al productiei coincide cu
                                  //  tokenul curent
                    tok = getNextToken(); // avansam la token-ul urmator
           else  A(); // a aparut o eroare; incercam o alta productie pentru A
           }
        else  if ( A == S ) reject();  // am analizat toate alternativele pentru S
              else ;
      } // end_while
} // end_A()
```

Observăm că acest algoritm folosește backtracking. Însă, pentru anumite tipuri de gramatici, și anume gramaticile de tip LL(1), pe care le vom studia în continuare, alegerea A-producției se poate realiza în mod unic. Dacă lucrăm cu o gramatică simbolică și $|z| = n,\ z = a_1 a_2 \dots a_n,\ a_{n+1} = \#$, atunci getNextToken constă din incrementarea unui indice k, inițializat cu 0, care indică caracterul curent din z.
Programul principal va consta din:

void main() {
// initializari; daca se lucreaza cu un fisier care contine programul sursa, atunci se
// se initializeaza pointerul curent pe primul caracter al fisierului; altfel $k = 0$;
$tok = getNextToken()$;
$S()$;
if $(tok \neq \#)$ reject(); // sau if $(tok \neq EOF)$
}

Totuși, metoda backtracking este foarte rar folosită pentru parsarea limbajelor de programare, deoarece nu este eficientă.

GRAMATICI RECURSIVE LA STÂNGA.

Definiție. Spunem că gramatica independentă de context $G=(N, \Sigma, S, P)$ este recursivă la stânga dacă în G există o derivare de forma $A \overset{+}{\Rightarrow} A\alpha$ pentru un neterminal A.
Spunem că neterminalul A prezintă recursivitate la stânga imediată dacă în G există o producție de forma $A \rightarrow A\alpha$.

ELIMINAREA RECURSIVITĂȚII LA STÂNGA IMEDIATE

Fie producțiile

$$A \rightarrow A\alpha_1 | A\alpha_2 | \cdots | A\alpha_m | \beta_1 | \cdots | \beta_n, m, n \geq 1 \quad (1)$$

unde niciun β_i nu începe cu $A, \alpha_j \neq \lambda$.
Înlocuim producțiile de mai sus cu:

$$A \rightarrow \beta_1 A' | \beta_2 A' | \cdots | \beta_n A'$$
$$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \cdots | \alpha_m A' | \lambda$$

Producțiile de mai sus generează aceleași șiruri ca producțiile din secvența (1), dar fără recursivitate la stânga imediată. Acest procedeu elimină recursivitatea la stânga imediată, dar nu și recursivitatea la stânga la modul general, ca în gramatica cu producțiile

$$S \rightarrow Aa | b$$
$$A \rightarrow Ac | Sd | \lambda$$

pentru care avem derivarea $S \Rightarrow Aa \Rightarrow Sda$.

ELIMINAREA RECURSIVITĂȚII LA STÂNGA

Algoritmul următor elimină recursivitatea la stânga pentru o gramatică independentă de context care nu are cicluri (derivări de forma $A \overset{+}{\Rightarrow} A$) și nici λ-producții (există algoritmi pentru eliminarea acestora dintr-o gramatică independentă de context).

INPUT: Gramatica independentă de context $G=(N, \Sigma, S, P)$ fără cicluri și fără λ-producții, cu $N = \{A_1, \cdots, A_n\}, A_1 = S$
OUTPUT: O gramatică echivalentă cu G, nerecursivă la stânga

for $(i = 1, \dots, n)\{$
    for $(j = 1, \dots, i-1)\{$
        inlocuieste fiecare productie de forma $A_i \to A_j\gamma$ cu productiile
        $A_i \to \delta_1\gamma|\delta_2\gamma|\cdots|\delta_k\gamma$, unde $A_j \to \delta_1|\delta_2|\cdots|\delta_k$ sunt toate A_j-productiile
    }
    elimina recursivitatea la stanga imediata pentru A_i
}

Observații. 1) Eliminarea recursivității imediate pentru un neterminal A introduce o λ-producție, însă pentru neterminalul nou introdus, A', pentru care nu mai există recursivitate la stânga.
2) Condiția ca gramatica să nu aibă λ-producții a fost impusă pentru a ne asigura că prin introducerea unei noi producții de tipul $A_i \to \delta_p\gamma$ unde $A_p \to \delta_p \in P, \delta_p \neq \lambda$, nu se introduce o recursivitate indirecta. Daca, în particular, nu este întâlnită o astfel de situație, atunci algoritmul funcționează și pentru gramatici cu λ-producții.

Factorizarea stângă a unei gramatici independente de context

Să considerăm o gramatică independentă de context $G=(N, \Sigma, S, P)$ în care există producțiile $A \to \alpha\beta, A \to \alpha\gamma$, unde β și γ încep cu simboluri diferite. Atunci când într-un parser de tip top down se pune problema alegerii între aceste două producții pe baza simbolurilor din intrare și a simbolurilor care apar în membrii drepți ai acestor producții, putem „amâna" această alegere în felul următor: înlocuim cele două producții de mai sus cu $A \to \alpha A', A' \to \beta|\gamma$, unde A' este un neterminal nou.

INPUT: Gramatica independentă de context $G=(N, \Sigma, S, P)$
OUTPUT: O gramatică echivalentă, factorizată la stânga

METODA. Pentru fiecare neterminal A se găsește cel mai lung prefix netrivial, $\alpha \neq \lambda$, pentru cel puțin două producții. Toate A-producțiile

$$A \rightarrow \alpha\beta_1|\alpha\beta_2|\cdots|\alpha\beta_m|\gamma$$

unde $A \rightarrow \gamma$ reprezintă toate A-producțiile care nu încep cu α, vor fi înlocuite cu

$$A \rightarrow \alpha A'|\gamma$$
$$A' \rightarrow \beta_1|\beta_2|\cdots|\beta_m$$

Se repetă metoda de mai sus până când nu mai există două producții care au în membrul stâng același neterminal și în membrul drept un șir care începe cu un prefix netrivial, comun celor două producții.

EXEMPLU Fie gramatica cu producțiile

$$S \rightarrow iEtSeS|iEtS|a$$
$$E \rightarrow b$$

care simbolizează cele două alternative pentru instrucțiunea ***if*** . După factorizarea stângă, producțiile gramaticii vor fi

$$S \rightarrow iEtSA'|a,$$
$$A' \rightarrow eS|\lambda$$
$$E \rightarrow b$$

Mulțimile FIRST și FOLLOW

Atât parserele de tip top-down cât și cele de tip bottom-up folosesc două tipuri de mulțimi definite astfel:

Definiție. Fie G=(N, Σ, S, P) o gramatică independentă de context, $\alpha \in (N \cup \Sigma)^*$ un cuvânt, $k \geq 1$. Definim mulțimile $FIRST_k(\alpha)$ și $FOLLOW_k(\alpha)$ prin:

$$FIRST_k(\alpha) = \{x \in \Sigma^* | (|x| < k \;\&\&\; \alpha \overset{*}{\Rightarrow}_s x) \text{ sau } (|x| = k \;\&\&\; \alpha \overset{*}{\Rightarrow}_s x\alpha', \alpha' \in (N \cup \Sigma)^*)\}$$

$$FOLLOW_k(\alpha) = \{x \in \Sigma^* | \exists\, S \overset{*}{\Rightarrow}_s w\alpha\beta, w \in \Sigma^*, \beta \in (N \cup \Sigma)^*, x \in FIRST_k(\beta)\}$$

În mulțimea $FIRST_k(\alpha)$ avem toate șirurile terminale cu lungimea mai mică decât k care pot fi derivate cu derivări stângi ce pornesc din α si toate șirurile terminale de lungime egală cu k care sunt prefixe ale unor șiruri ce provin din α printr-o derivare stângă.

Observăm că pentru $\alpha \in \Sigma^*$, $FIRST_k(\alpha)$ se reduce la un singur șir. De asemenea, pentru un limbaj $L \subseteq \Sigma^*$ avem

$$FIRST_k(L) = \bigcup_{\alpha \in L} FIRST_k(\alpha)$$

Proprietăți ale mulțimilor FIRST și FOLLOW

Fie G=(N, Σ, S, P) o gramatică independentă de context, $\alpha, \beta \in (N \cup \Sigma)^*$, $k \geq 1$. Din definiția mulțimilor FIRST și FOLLOW rezultă:

- $FIRST_k(\alpha\beta) = FIRST_k\big(FIRST_k(\alpha)FIRST_k(\beta)\big)$
- Dacă $\alpha \overset{\star}{\Rightarrow} \beta$ atunci $FIRST_k(\beta) \subseteq FIRST_k(\alpha)$
- Dacă $A \to \alpha \in P$ atunci $FIRST_k(\alpha) \subseteq FIRST_k(A)$
- Pentru $k = 0, FIRST_0(\alpha) = \{\lambda\}, \forall \alpha \in (N \cup \Sigma)^*$

Algoritm pentru determinarea mulțimilor FIRST

INPUT: Gramatica independentă de context G=(N, Σ, S, P), $k \geq 1$
OUTPUT: Multimile $FIRST_k(X), X \in N \cup \Sigma$

Inițializare.
$FIRST_k(\lambda) = \{\lambda\}$
$FIRST_k(X) = \{X\}, \forall X \in \Sigma$
Pentru $X \to X_1 \cdots X_m, m \geq 0, X_1, \cdots, X_m \in \Sigma$,
$FIRST_k(X) \leftarrow \{X_1 \cdots X_m\}$ daca $m \leq k$ sau
$FIRST_k(X) \leftarrow \{X_1 \cdots X_k\}$ daca $m > k$
Repeta cat timp se adauga noi siruri
$\forall A \to X_1 \cdots X_m \in P, m \geq 1, |X_1 \cdots X_m|_N \geq 1$, astfel că $FIRST_k(X_1) \neq \emptyset$, ..., $FIRST_k(X_m) \neq \emptyset$, atunci **adauga** la $FIRST_k(A)$ multimea
$FIRST_k\big(FIRST_k(X_1) \ldots FIRST_k(X_m)\big)$

Pentru un sir $\alpha = Z_1 \ldots Z_m, Z_1, \ldots, Z_m \in N \cup \Sigma$,
avem $FIRST_k(\alpha) = FIRST_k\big(FIRST_k(Z_1) \ldots FIRST_k(Z_m)\big)$

Algoritm pentru determinarea mulțimilor FOLLOW

INPUT: Gramatica independentă de context G=(N, Σ, S, P), $k \geq 1$
OUTPUT: Multimile $FOLLOW_k(A), A \in N$

Inițializare.
$FOLLOW_k(S) \leftarrow \{\lambda\}$

Repeta cat timp se adauga noi siruri

$\forall A \to \alpha B \beta \in P, B \in N, \alpha, \beta \in (N \cup \Sigma)^*$,

la $FOLLOW_k(B)$ **adauga** multimea $FIRST_k(\beta \cdot FOLLOW_k(A))$

Observații. 1) $FIRST_k(\beta \cdot FOLLOW_k(A))$ este un limbaj finit.
2) Pentru o productie $A \to \alpha \in P, |\alpha|_N \geq 2$, se iau in considerare toți neterminalii care apar în α.

GRAMATICI ȘI LIMBAJE DE TIP $LL(k)$

Gramaticile de tip $LL(k)$ sunt gramatici neambigue, pentru care se pot implementa algoritmi de parsare liniari de tip top-down. Denumirea acestora provine din *Parsing from* **L***eft to Right using* **L***eftmost derivations and* **k** *symbols lookahead.* Intuitiv, o gramatică este de tip $LL(k)$ dacă atunci când se ajunge într-un punct în care algoritmul trebuie să ia o decizie în privința producției care urmează să se aplice pentru neterminalul A, 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 $LL(k)$ dacă pentru orice două derivări stângi:

$$S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha \underset{s}{\Rightarrow} w\beta\alpha \overset{*}{\underset{s}{\Rightarrow}} wx \quad \text{și} \quad S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha \underset{s}{\Rightarrow} w\gamma\alpha \overset{*}{\underset{s}{\Rightarrow}} wy$$

unde $S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha$ este porțiunea comună a celor două derivări, $w, x, y \in \Sigma^*$, $\alpha \in (N \cup \Sigma)^*, A \to \beta, A \to \gamma \in P$, astfel încât $FIRST_k(x) = FIRST_k(y)$, **atunci** $\beta = \gamma$.

Cu alte cuvinte, după ce parserul a analizat prefixul w și trebuie în continuare să aleagă o producție pentru A, atunci pe baza următoarelor k simboluri din intrare (furnizate de $FIRST_k(x) = FIRST_k(y)$) se poate deduce care este acea producție, (unică, dacă există) care se poate aplica astfel încât să poată fi generate în continuare următoarele $FIRST_k(x)$ simboluri din intrare.

Definiție. Spunem că gramatica G este de tip LL dacă există $k \geq 0$ pentru care G este de tip $LL(k)$.
Spunem că un limbaj L este de tip $LL(k)$ dacă există o gramatică independentă de context de tip $LL(k)$ care îl generează.

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

Exemple.

1. $S \to aAa|bBb$, $A \to aAa|c$, $B \to bBb|c$ — Aceasta este o gramatică de tip $LL(1)$ deoarece fiecare dintre cele două producții ale neterminalilor S, A, B încep cu simboluri diferite, ceea ce înseamnă că este necesară examinarea unui singur simbol *lookahead*
2. $S \to aAa$, $A \to aAa|\lambda$ — Această gramatică nu este de tip $LL(k)$ pentru niciun *k*.

Într-adevăr, avem derivările stângi

$$S \underset{s}{\overset{*}{\Rightarrow}} a^k A a^k \Rightarrow a^{k+1} A a^{k+1} \Rightarrow a^{k+1} a^{k+1} = a^k a^{k+2}$$

$$S \underset{s}{\overset{*}{\Rightarrow}} a^k A a^k \Rightarrow a^k a^k$$

și $FIRST_k(a^k a^{k+2}) = FIRST_k(a^k a^k) = \{a^k\}$, dar după primii *k* pași ai celor două derivări s-au aplicat derivări distincte, $A \to aAa$ și $A \to \lambda$, deci *G* nu este $LL(k)$

CÂTEVA PROPRIETĂȚI DE BAZĂ ALE GRAMATICILOR DE TIP $LL(k)$

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

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

$$S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha \underset{s}{\Rightarrow} w\beta\alpha \underset{s}{\overset{*}{\Rightarrow}} wx = z \quad \text{și} \quad S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha \underset{s}{\Rightarrow} w\gamma\alpha \underset{s}{\overset{*}{\Rightarrow}} wy = z$$

unde $S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha$ este porțiunea comună a celor două derivări (posibil ca $S = wA\alpha$), iar $A \to \beta$ și $A \to \gamma$ sunt două producții distincte ($\beta \neq \gamma$). Dar, deoarece $wx = z = wy$, rezultă că $x = y$, deci $FIRST_k(x) = FIRST_k(y)$, și, deoarece *G* este $LL(k)$, rezultă că $\beta = \gamma$, contradicție.

Rezultă că *G* nu este ambiguă q.e.d.

Propoziția 2. Orice gramatică recursivă la stânga fără simboluri inutilizabile nu este de tip LL.

Demonstrație. Exercițiu 1 (Simbolurile inutilizabile sunt acele simboluri terminale sau neterminale care nu apar în derivarea niciunui șir din limbajul gramaticii. Din acest motiv, acestea pot fi eliminate, împreună cu toate producțiile în care apar.)

Propoziția 3 (teorema de caracterizare). Fie $G=(N, \Sigma, S, P)$ o gramatică independentă de context fără simboluri inutilizabile. Atunci G este de tip $LL(k)$ dacă și numai dacă pentru orice derivare stângă $S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha$, $w \in \Sigma^*$, și pentru orice două producții distincte $A \to \beta$ și $A \to \gamma$ $(\beta \neq \gamma)$, avem

$$FIRST_k(\beta\alpha) \cap FIRST_k(\gamma\alpha) = \emptyset.$$

Demonstrație. Exercițiu 2

GRAMATICI ȘI LIMBAJE DE TIP $LL(k)$ TARI

Definiție. Fie $G=(N, \Sigma, S, P)$ este o gramatică independentă de context fără simboluri inutilizabile și $k \geq 0$ dat. Spunem că G este de tip $LL(k)$ tare dacă pentru orice două producții distincte $A \to \beta$ și $A \to \gamma$, $\beta \neq \gamma$, avem

$$FIRST_k(\beta \cdot FOLLOW_k(A)) \cap FIRST_k(\gamma \cdot FOLLOW_k(A)) = \emptyset$$

Exemple.

3. $S \to aAaa|aBba$
   $A \to b|\lambda$
   $B \to c$

Aceasta este o gramatică de tip $LL(2)$ tare. Avem
$FOLLOW_2(S) = \{\lambda\}, FOLLOW_2(A) = \{aa\}$ și pentru productiile lui S
$FIRST_2(aAaa \cdot FOLLOW_2(S)) = \{ab, aa\}$, $FIRST_2(aBba \cdot FOLLOW_2(S)) = \{ac\}$, și $\{ab, aa\} \cap \{ac\} = \emptyset$, iar pentru productiile lui A
$FIRST_2(b \cdot FOLLOW_2(A)) = \{ba\}$, $FIRST_2(\lambda \cdot FOLLOW_2(A)) = \{aa\}$, și $\{ba\} \cap \{aa\} = \emptyset$. Rezultă că gramatica este $LL(2)$ tare.

4. $S \to aAaa|aAba$
   $A \to b|\lambda$

Exercițiu 3. Gramatica de mai sus este de tip $LL(2)$ dar nu este $LL(2)$ tare.

Propoziția 4. Orice gramatică de tip $LL(k)$ tare este $LL(k)$.

Demonstrație. Fie $G=(N, \Sigma, S, P)$ o gramatică independentă de context de tip $LL(k)$ tare, fără simboluri inutilizabile. Presupunem că G nu este de tip $LL(k)$.

Atunci, din Propoziția 2 rezultă că există în G o derivare stângă $S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha$, $w \in \Sigma^*$, și există două producții distincte $A \to \beta$ și $A \to \gamma$ $(\beta \neq \gamma)$, pentru care $FIRST_k(\beta\alpha) \cap FIRST_k(\gamma\alpha) \neq \emptyset$. Fie $z \in FIRST_k(\beta\alpha) \cap FIRST_k(\gamma\alpha), z \in \Sigma^*$. Atunci, din definiția mulțimilor $FIRST_k$ rezultă că avem în G derivările: $\beta\alpha \overset{*}{\underset{s}{\Rightarrow}} zu$ și $\gamma\alpha \overset{*}{\underset{s}{\Rightarrow}} zv$, $u, v \in (N \cup \Sigma)^*, |z| \leq k$. Daca $|z| < k$, atunci $u = v = \lambda$. Deoarece G nu are simboluri inutilizabile, rezulta ca exista in G o derivare care porneste din S a unei forme sententiale în care apare A, $S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha$, care conduce la derivarea unui șir terminal. Atunci în G avem derivările: $S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha \underset{s}{\Rightarrow} w\beta\alpha \overset{*}{\underset{s}{\Rightarrow}} wzu$ și $S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha \underset{s}{\Rightarrow} w\gamma\alpha \overset{*}{\underset{s}{\Rightarrow}} wzv$, iar $FIRST_k(zu) = FIRST_k(zv) = \{z\}$. Din definiția gramaticilor de tip $LL(k)$, rezultă că $\beta = \gamma$, contradicție.

Reciproca acestei afirmații nu este întotdeauna valabilă, așa cum am văzut în exemplul 4. Însă, pentru $k = 1$ vom arata ca este adevărată.

Observație. Pentru $k = 1$ vom scrie $FIRST$ și $FOLLOW$ în loc de $FIRST_1$ și $FOLLOW_1$.

Propoziția 5. Fie $G=(N, \Sigma, S, P)$ o gramatică independentă de context fără simboluri inutilizabile. Atunci G este $LL(1)$ dacă și numai dacă este $LL(1)$ tare.

Demonstrație. „⇒” Fie $G=(N, \Sigma, S, P)$ o gramatică independentă de context fără simboluri inutilizabile de tip $LL(1)$. Presupunem că G nu este $LL(1)$ tare. Din definița gramaticilor LL tari rezultă că există două producții distincte $A \to \beta$, $A \to \gamma$, $\beta \neq \gamma$ cu $FIRST(\beta \cdot FOLLOW(A)) \cap FIRST(\gamma \cdot FOLLOW(A)) \neq \emptyset$. Fie $x \in FIRST(\beta \cdot FOLLOW(A)) \cap FIRST(\gamma \cdot FOLLOW(A)), x \in \Sigma \cup \{\lambda\}$. Avem cazurile:

Cazul 1. $x \in \Sigma, x \in FIRST(\beta) \cap FIRST(\gamma)$.

Rezultă că avem în G derivările: $\beta \overset{*}{\underset{s}{\Rightarrow}} xu$ și $\gamma \overset{*}{\underset{s}{\Rightarrow}} xv$, $u, v \in (N \cup \Sigma)^*$. Deoarece G nu are simboluri inutilizabile, rezulta ca exista in G o derivare care porneste din S a unei forme sententiale în care apare A, $S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha$, care conduce la derivarea unui șir terminal. Atunci în G avem derivările $S \overset{*}{\underset{s}{\Rightarrow}} wA\alpha \overset{*}{\underset{s}{\Rightarrow}} w\beta\alpha \overset{*}{\underset{s}{\Rightarrow}} wxu \overset{*}{\underset{s}{\Rightarrow}} wxu' \in \Sigma^*$ și

$S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha \underset{s}{\overset{*}{\Rightarrow}} w\gamma\alpha \underset{s}{\overset{*}{\Rightarrow}} wxv \underset{s}{\overset{*}{\Rightarrow}} wxv' \in \Sigma^*$, iar $FIRST(xu') = FIRST(xv') = \{x\}$.
Din definiția gramaticilor de tip $LL(1)$, rezultă că $\beta = \gamma$, contradicție.
Cazul 2. $x \in \Sigma, x \in FIRST(\beta) \cap FOLLOW(A), \gamma \underset{s}{\Rightarrow} \lambda$ (orice derivare în *G* o putem rescrie ca pe o derivare stângă).
Rezultă că avem în *G* derivările $\beta \underset{s}{\overset{*}{\Rightarrow}} xu, u \in (N \cup \Sigma)^*$ și $S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha, x \in FIRST(\alpha)$, adica exista o derivare $\alpha \underset{s}{\overset{*}{\Rightarrow}} xv$. Rezulta ca în *G* avem derivările
$S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha \Rightarrow w\beta\alpha \underset{s}{\overset{*}{\Rightarrow}} wxu$ și
$S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha \Rightarrow w\gamma\alpha \underset{s}{\overset{*}{\Rightarrow}} w\alpha \underset{s}{\overset{*}{\Rightarrow}} wxv$, iar $FIRST(xu) = FIRST(xv) = \{x\}$. Din definiția gramaticilor de tip $LL(1)$, rezultă că $\beta = \gamma$, contradicție.
Cazul 3. $x \in \Sigma, x \in FIRST(\gamma) \cap FOLLOW(A), \beta \underset{s}{\Rightarrow} \lambda$. Este analog cazului 2.
Cazul 4. $x = \lambda, x \in FOLLOW(A), \beta \underset{s}{\Rightarrow} \lambda, \gamma \underset{s}{\Rightarrow} \lambda$.
Rezulta ca exista in *G* o derivare $S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha$, cu $\lambda \in FIRST(\alpha)$, deci $\alpha \underset{s}{\Rightarrow} \lambda$. Atunci în *G* avem derivările $S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha \Rightarrow w\beta\alpha \underset{s}{\overset{*}{\Rightarrow}} w\alpha \underset{s}{\overset{*}{\Rightarrow}} w = w\lambda,$
$S \underset{s}{\overset{*}{\Rightarrow}} wA\alpha \Rightarrow w\gamma\alpha \underset{s}{\overset{*}{\Rightarrow}} w\alpha \underset{s}{\overset{*}{\Rightarrow}} w = w\lambda,$ cu $FIRST(\lambda) = FIRST(\lambda) = \{\lambda\}$. Din definiția gramaticilor de tip $LL(1)$, rezultă că $\beta = \gamma$, contradicție.
„⇐” Rezulta din Propozitia 4.

O consecință foarte importantă a acestei teoreme o vom vedea în cursul următor, în care vom studia algoritmul *k*-predictiv pentru gramatici $LL(k)$ tari. Precizăm, totodată, că majoritatea limbajelor de programare a căror sintaxă se bazează pe gramatici de tip *LL*, au de fapt la bază o gramatică sintactică de tip $LL(1)$.

Cel mai simplu algoritm sintactic liniar este algoritmul recursiv descendent pentru gramatici de tip $LL(1)$, care este o particularizare a algoritmului recursiv descendent general prezentat mai sus, în care am folosit backtracking.
În cazul algoritmului recursiv descendent pentru gramatici de tip $LL(1)$, alegerea unei producții pentru un neterminal *A* care are producțiile $A \to \alpha_1, \dots, A \to \alpha_n$ se face pe baza multimilor $FIRST(\alpha_i \cdot FOLLOW(A)), i = 1 \dots n$, care sunt disjuncte două câte două.

Algoritmul propriu-zis:

INPUT: gramatica independentă de context $G = (N, \Sigma, S, P)$ de tip $LL(1)$ și $w = z\#$, unde $z \in \Sigma^*$, # un simbol nou ce marcheaza finalul lui z ($\# = EOF$ dacă G formalizează sintaxa unui limbaj de programare; in acest caz Σ reprezintă mulțimea categoriilor de atomi lexicali (token-ii) acceptați de limbajul respectiv, iar EOF marchează sfârșitul fișerului care conține programul sursă, program reprezentat de z)
OUTPUT: derivarea stângă (unică, dacă există) a lui z, dacă $z \in L(G)$ sau un mesaj de eroare in caz contrar.

ALGORITM (parser recursiv descendent)

Fie tok o variabila globala in care retinem token-ul curent și getNextToken functia (scanner-ul) care furnizeaza următorul token. Pentru fiecare neterminal $A \in N$ cu producțiile $A \to \alpha_1, \dots, A \to \alpha_n$ se implementează funcția:

void A() {
    $i \leftarrow 0$;
    Determina $i, 1 \le i \le n$, pentru care $tok \in FIRST(\alpha_i \cdot FOLLOW(A))$;
    //daca exista, i este unic cu aceasta proprietate
    if ($i == 0$) error(); //sirul de intrare (programul sursa) nu poate fi parsat
    else {
        fie $A \to X_1 \cdots X_k$ cea de-a *i*-a productie a lui *A*;
        write(numarul de ordine al productiei, sau productia insasi)
        for ($i = 1, \dots, k$)
        { **if** (X_i este neterminal)
            X_i(); // se apeleaza functia corespunzatoare neterminalului X_i
            else if ($X_i == tok$)
                $tok =$ getNextToken(); //trecem la token-ul urmator
            else error(); // a aparut o eroare; token-ul curent nu coincide
                // cu simbolul introdus de productia lui *A*; ne oprim
        } // end_for
    } // end_if
} // and_A()

Programul principal va consta din:
void main() {
// initializari
$tok =$ getNextToken(); //se considera ca EOF (#) este in $follow(S)$
S();
if ($tok != EOF$) error();

}

EXERCIȚII

4) Caracterizați gramaticile și limbajele de tip $LL(0)$.
5) Arătați că pentru orice $k \geq 0$ există o gramatică independentă de context *G* care este $LL(k+1)$ dar nu este $LL(k)$.
!! 6) Arătați că pentru orice $k \geq 0$ există un limbaj independent de context *G* care este $LL(k+1)$ dar nu este $LL(k)$.
! 7) Arătați că limbajul determinist

$$L = \{a^n | n \geq 1\} \cup \{a^n b^n | n \geq 1\}$$

nu este $LL(k)$ pentru niciun k.