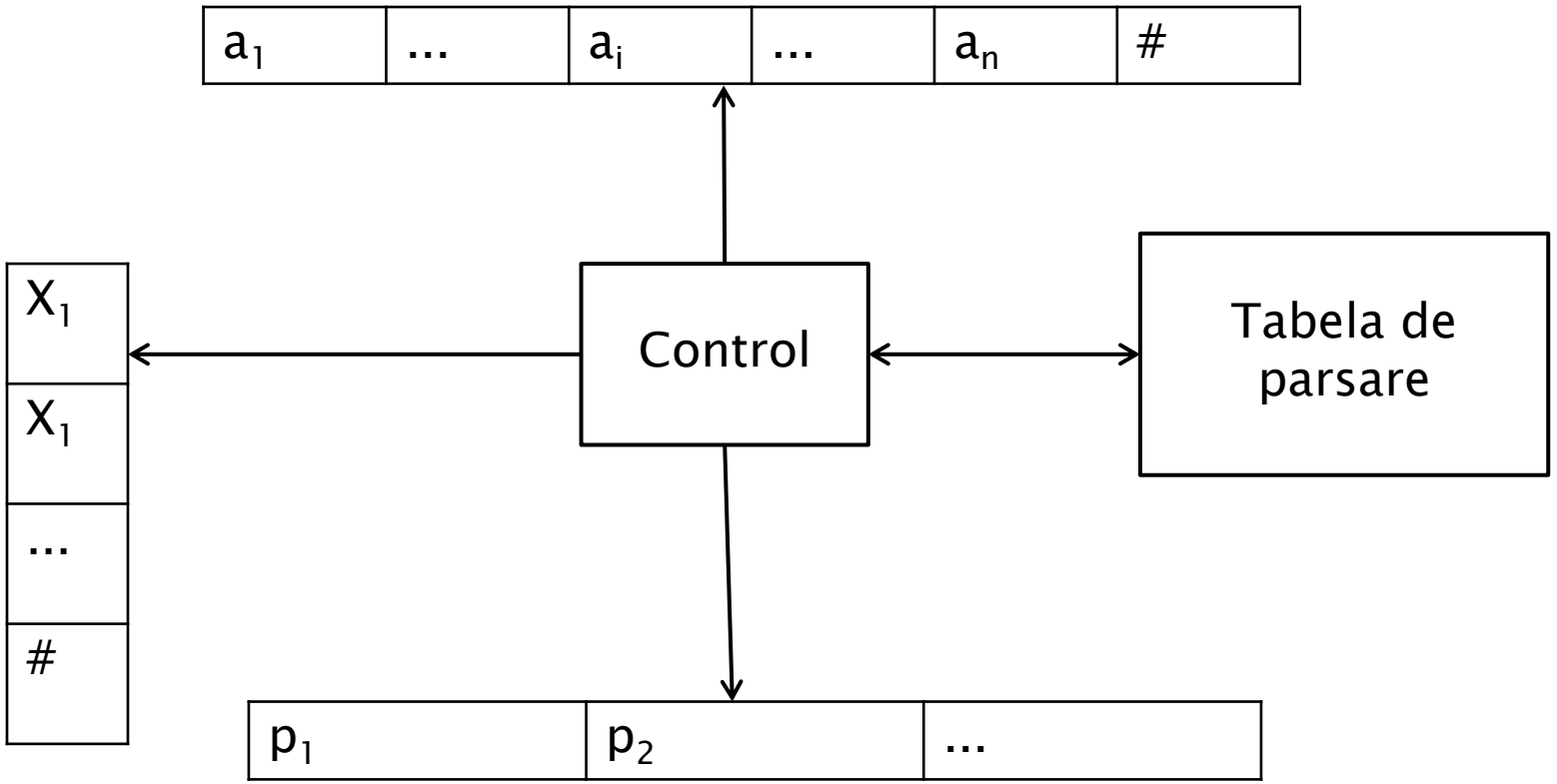
Limbaje formale, automate şi compilatoare

Curs 9-10

Recapitulare

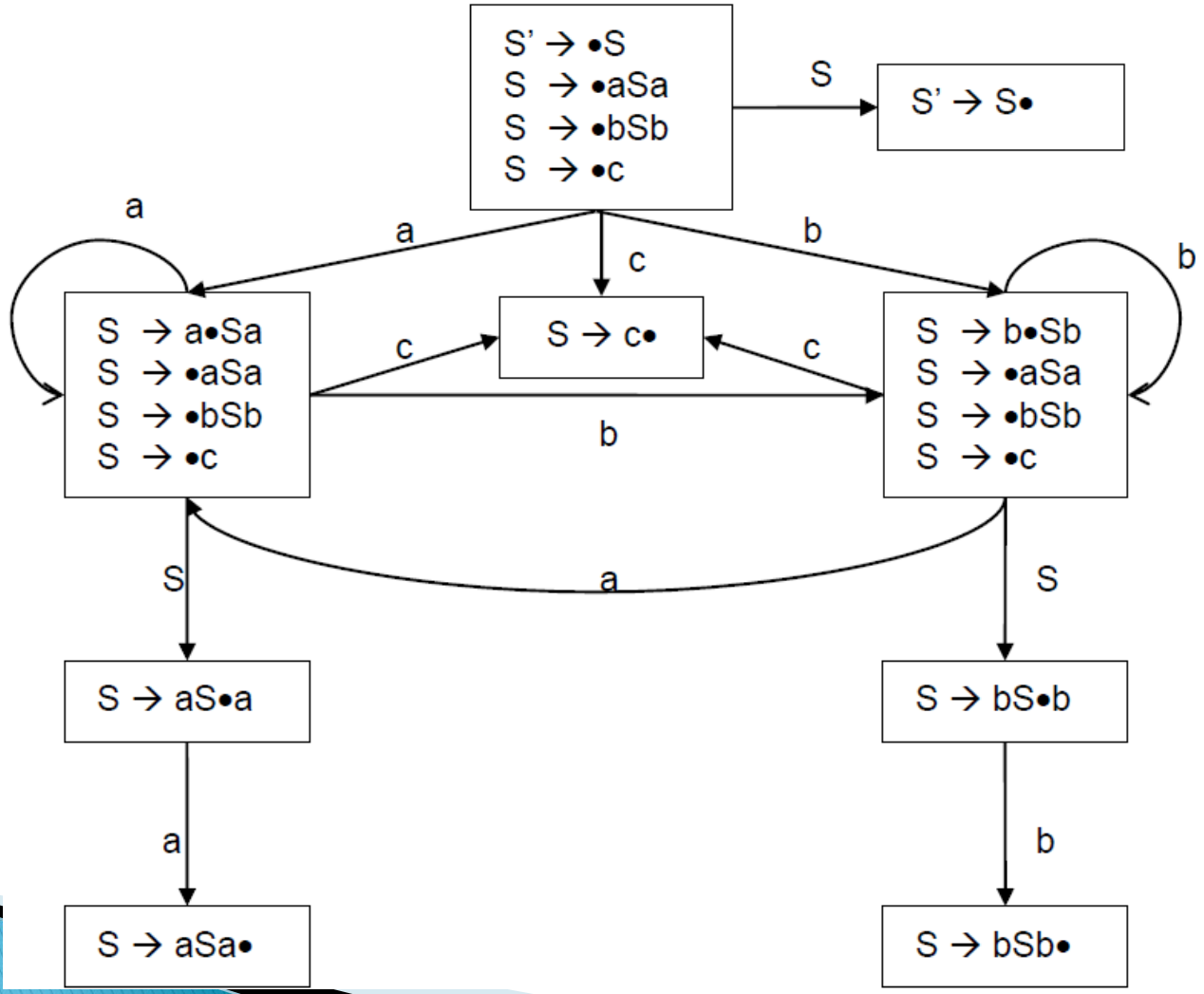
- Analiza sintactică ascendentă
  - Parser ascendent general
- Gramatici LR(k)
  - Definiţie
  - Proprietăți
- Gramatici LR(0)
  - Teorema de caracterizare LR(0)
  - Automatul LR(0)

Parser ascendent general



Automatul LR(0) – Exemplu

- S’ → S, S → aSa | bSb | c



Cuprins

- Analiza sintactică LR(0)
- Generatoare de analizoare sintactice
- Mulţimile FIRST, FOLLOW
- Gramatici SLR(1)
  - Tabela de parsare SLR(1)
  - Analiza sintactică SLR(1)
- Gramatici LR(1)

Algoritmul de analiză LR(0)

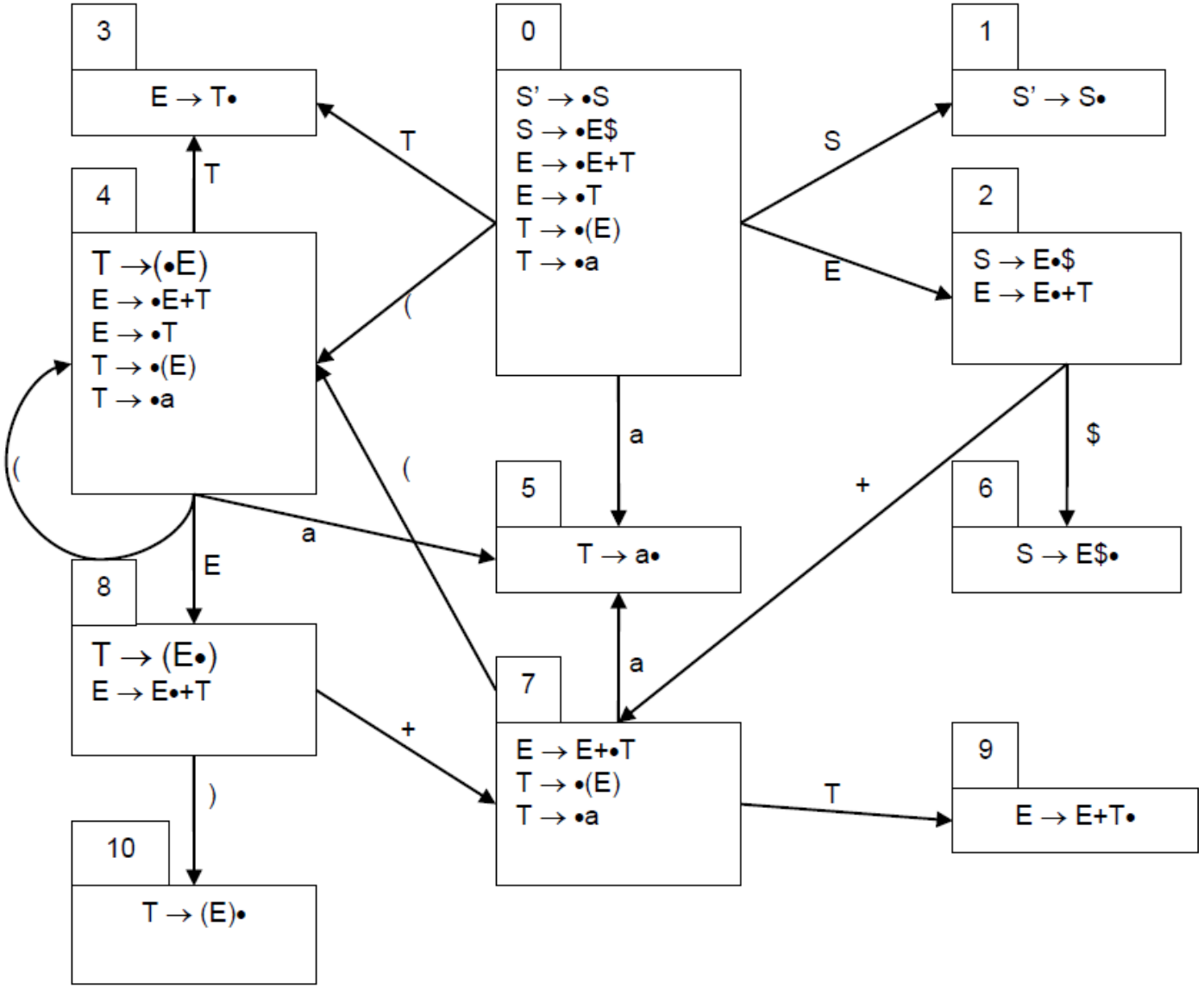
- Tabela de parsare coincide cu automatul LR(0), M.
- Configuraţie: $(\sigma, u\#, \pi)$ unde $\sigma \epsilon t_0 T^*$, $u \epsilon T^*$, $\pi \epsilon P^*$.
- Configuraţia iniţială este $(t_0, w\#, \varepsilon)$,
- Tranziţiile:
  - Deplasare: $(\sigma t, au\#, \pi) \vdash (\sigma tt', u\#, \pi)$ dacă $g(t, a) = t'$.
  - Reducere: $(\sigma t \sigma' t', u\#, \pi) \vdash (\sigma tt'', u\#, \pi r)$ dacă $A \rightarrow \beta\bullet \in t'$, $r = A \rightarrow \beta$, $|\sigma' t'| = |\beta|$ şi $t'' = g(t, A)$.
  - Acceptare: $(t_0 t_1, \#, \pi)$ este configuraţia de acceptare dacă $S' \rightarrow S\bullet \in t1$, π este parsarea acestuia.
  - Eroare: o configuraţie căreia nu i se poate aplica nici o tranziţie

Algoritmul de analiză LR(0)

- char ps[]= “w#”; //ps este sirul de intrare w
- i = 0; // pozitia in sirul de intrare
- STIVA.push(t0); // se initializeaza stiva cu t0
- while(true) { // se repeta pana la succes sau eroare
  - t = STIVA.top();
  - a = ps[i] // a este simbolul curent din intrare
  - if(g(t, a)≠Ø { //deplasare
    - STIVA.push(g(t, a));
    - i++; //se inainteaza in intrare
    - }
  - else {
  - if(A → $X_1X_2…X_m$• ε t){
    - if(A == „S“”)
      - if(a == „#“)exit(“acceptare”);
      - else exit(“eroare”);
    - else // reducere
      - for(i = 1; i □m; i++) STIVA.pop();
        - STIVA.push(g(top(STIVA), A));
    - } //endif
  - else exit(“eroare”);
  - }//endelse
- }//endwhile

Exemplu

- S’ → S S → E$ E → E+T T →(E) E → T T → a



Exemplu

- S’ → S S → E$ E → E+T T →(E) E → T T → a

Stiva	Intrare	Acţiune	Ieşire
0	a+(a+a)$#	deplasare	
05	+(a+a)$#	reducere	T → a
03	+(a+a)$#	reducere	E → T
02	+(a+a)$#	deplasare	
027	(a+a)$#	deplasare	
0274	a+a)$#	deplasare	
02745	+a)$#	reducere	T → a
02743	+a)$#	reducere	E → T
02748	+a)$#	deplasare	
027487	a)$#	deplasare	
0274875	)$#	reducere	T → a
0274879	)$#	reducere	E → E+T
02748	)$#	deplasare	
02748'10'	$#	reducere	T → (E)
0279	$#	reducere	E → E+T
02	$#	deplasare	
026	#	reducere	S → E$
01	#	acceptare	

Corectitudinea parserului LR(0)

- **Lema 1, 2** Fie G = (N, T, S, P) o gramatică LR(0), $t_0\sigma$, $t_0\tau$ drumuri în automatul LR(0) etichetate cu φ respectiv γ şi u, v ∈ T*. Atunci, dacă în parserul LR(0) are loc $(t_0\sigma, uv\#, \varepsilon) \vdash^+ (t_0\tau, v\#, \pi)$, atunci în G are loc derivarea $\varphi \Rightarrow_{dr\,\pi} u$ şi reciproc.

- **Teoremă** Dacă G este gramatică LR(0) atunci, oricare ar fi cuvântul de intrare w ∈ T*, parserul LR(0) ajunge la configuraţia de acceptare pentru w, adică $(t_0\sigma, uv\#, \varepsilon) \vdash^+ (t_0\tau, v\#, \pi)$ dacă şi numai dacă $\varphi \Rightarrow_{dr\,\pi} u$

Analiza sintatică – generatoare automate

- **YACC** (“Yet Another Compiler Compiler’’) conceput şi realizat în 1975 de S. C. Johnson la Bell Laboratory
- Bison, compatibil în întregime cu YACC, scris de Robert Corbett şi Richard Stallmann

Compilarea cu YACC

- `yacc <nume_fişier>`
- Rezultatul este un fişier C

- Structura fişierului yacc
  - Declaraţii C şi yacc
  - %%
  - Reguli (gramatică de tipul 2)
  - %%
  - Cod C

Sintaxa yacc – declaraţii

- %{ Declarații C %}
- Declararea de simboluri terminale:
  - %token <terminal>
- Declararea simbolului de start al gramaticii:
  - %start <neterminal>
- Alte declarații:
  - *type* – definirea tipului;
  - *left* – asociativitate stânga a operatorilor;
  - *right* – asociativitate dreapta a operatorilor;
  - *prec* – precedenţa operatorilor;
  - *nonassoc* – neasociativitate

Sintaxa yacc – reguli

- neterminal: <parte dreaptă> {cod C}

```
expr:NUMAR{$$=$1;}
     |expr "+"expr {$$=$1+$3;}
     |expr "-" expr{$$=$1-$3;}
     |expr "*" expr{$$=$1*$3;}
     |expr "/" expr{$$=$1/$3;}
     |"("expr")" {$$=$2;}
;
```

Gramatici SLR(1)

- **Definiţie**
  - Fie G o gramatică pentru care automatul LR(0) conţine stări inconsistente (deci G nu este LR(0)). Gramatica G este gramatică SLR(1)dacă oricare ar fi starea t a automatului LR(0) sunt îndeplinite condiţiile:

  - -Dacă $A \rightarrow \alpha\bullet$, $B \rightarrow \beta \bullet \in$ t atunci FOLLOW(A) $\cap$FOLLOW(B) $= \varnothing$;
  - -Dacă $A \rightarrow \alpha\bullet$, $B \rightarrow \beta \bullet a\gamma \in$ t atunci a $\notin$ FOLLOW(A).

- Analiza sintactică SLR(1) este similară cu cea LR(0);tabela de analiză sintactică are două componente:
  - -Prima, numită ACŢIUNE, determină dacă parserul va face deplasare respectiv reducere, în funcţie de starea ce se află în topul stivei şi de simbolul următor din intrare
  - -Cea de a doua, numită GOTO, determină starea ce se va adăuga în stivă în urma unei reduceri.

Mulţimile FIRST şi FOLLOW

- FIRST(α) = {a|a ∈ T, $\alpha \underset{st}{\Rightarrow}^* au$ } ∪
  if ($\alpha \underset{st}{\Rightarrow}^* \varepsilon$) then {$\varepsilon$} else ∅.

- FOLLOW(A) = {a|a ∈ T ∪ {ε}, $S \underset{st}{\Rightarrow}^* uA\gamma$,
  a ∈ FIRST (γ) }

Determinare FIRST

- 1.for(X $\in$ Σ)
  - 2.if(X $\in$ T)FIRST(X)={X} else FIRST(X)=$\varnothing$;
- 3.for(A$\rightarrow$aβ $\in$ P)
  - 4.FIRST(A)=FIRST(A)$\cup${a};
- 5.FLAG=true;
- 6.while(FLAG){
  - 7.FLAG=false;
  - 8.for(A $\rightarrow$ $X_1X_2\ldots X_n$ $\in$ P){
    - 9.i=1;
    - 10.if((FIRST(X1) $\notin$ FIRST(A)){
      - 11.FIRST(A)=FIRST(A) $\cup$(FIRST(X1);
      - 12.FLAG=true;
    - 13.}//endif
    - 14.while(i<n&&X_i $\Rightarrow^*_{st}$ ε)
      - 15.if((FIRST(X_{i+1}) $\notin$ FIRST(A)){
        - 16.FIRST(A)=FIRST(A)$\cup$ FIRST(X_{i+1});
        - 17.FLAG=true;i++;
      - }//endif
    - }//endwhile
  - }//endfor
- }//endwhile
- for(A $\in$ N)
  - if(A_{st} $\Rightarrow^*$ ε)FIRST(A)=FIRST(A) $\cup${ε};

Determinare FIRST

- **Intrare:** Gramatica G=(N,T,S,P).
- Mulţimile FIRST(X),X $\in$ Σ.
- $\alpha = X_1X_2\ldots X_n$, $X_i \in \Sigma, 1 \leq i \leq n$.
- **Ieşire:** FIRST(α).

- 1.FIRST(α)=FIRST(X_1)-{ε};i=1;
- 2.while(i<n && $X_i \Rightarrow^+ \varepsilon$){
  - 3.FIRST(α)=FIRST(α)$\cup$(FIRST(X_{i+1})-{ε});
  - 4.i=i+1;
- }//endwhile
- 5.if(i==n && $X_n \Rightarrow^+ \varepsilon$)
  - 6.FIRST(α)=FIRST(α) $\cup$ {ε};

Exemplu

- Fie gramatica:
- S → E | B, E → ε, B → a | beginSC end, C → ε | ; SC

- FIRST(S) = {a, begin, ε} FIRST(E) = {ε}
- FIRST(B) = {a, begin} FIRST(C) = {;, ε}.
- FIRST(SEC) = {a, begin, ;, ε},
- FIRST(SB)= {a, begin},
- FIRST(;SC)= {;}.

Determinarea FOLLOW

- $\varepsilon \in$ FOLLOW(S).
- Dacă $A \rightarrow \alpha B \beta X \gamma \in P$ şi $\beta \Rightarrow^{+} \varepsilon$, atunci FIRST(X) – $\{\varepsilon\} \subseteq$ FOLLOW (B).
  - $S \Rightarrow^{*} \alpha_1 A \beta_1 \Rightarrow \alpha_1 \alpha B \beta X \gamma \beta_1 \Rightarrow^{*} \alpha_1 \alpha B X \gamma \beta_1$ şi atunci rezultă FIRST(X)–$\{\varepsilon\} \subseteq$ FOLLOW (B).
- Dacă $A \rightarrow \alpha B \beta \in P$ atunci FIRST(β)–$\{\varepsilon\} \subseteq$ FOLLOW (B).
- Dacă $A \rightarrow \alpha B \beta \in P$ şi $\beta \Rightarrow^{+} \varepsilon$, atunci FOLLOW(A) $\subseteq$ FOLLOW(B).

Determinarea FOLLOW

- 1. for(A $\in$ Σ)FOLLOW(A)=$\varnothing$;
- 2.FOLLOW(S)={ε};
- 3.for(A $\rightarrow$ $X_1X_2\ldots X_n$){
- 4.i=1;
  - 5.while(i<n){
    - 6.while($X_i \notin N$)++i;
    - 7.if(i<n){
      - 8.FOLLOW(Xi)= FOLLOW(X_i)$\cup$ (FIRST($X_{i+1}X_{i+2}\ldots X_n$)-{ε});
      - 9.++i;
    - }//endif
  - }//endwhile
- }//endfor

Determinarea FOLLOW

- 10.FLAG=true;
- 11.while(FLAG){
  - 12.FLAG=false;
  - 13.for(A → $X_1X_2…X_n$){
    - 14.i=n;
    - 15.while(i>0 && $X_i \in N$){
      - 16.if(FOLLOW(A) $\not\subset$ FOLLOW(X_i)){
        - 17.FOLLOW(Xi)=FOLLOW(X_i) $\cup$ FOLLOW(A);
        - 18.FLAG=true;
      - 19.}//endif
      - 20.if($X_i \Rightarrow^+ \varepsilon$)--i;
      - 21.else continue;
    - 22.}//endwhile
  - 23.}//endfor
- 24.}//endwhile

Exemplu

- Fie gramatica:
- S → E | B, E → ε, B → a | begin SC end,
  C → ε | ; SC

- FOLLOW(S)=FOLLOW(E)=FOLLOW(B) ={ε, ; , end}
- FOLLOW(C) = {end}.

Gramatici SLR(1)

- **Definiţie**
  - Fie G o gramatică pentru care automatul LR(0) conţine stări inconsistente (deci G nu este LR(0)). Gramatica G este gramatică SLR(1)dacă oricare ar fi starea t a automatului LR(0) sunt îndeplinite condiţiile:

  - -Dacă $A \rightarrow \alpha\bullet$, $B \rightarrow \beta \bullet \in$ t atunci FOLLOW(A) $\cap$FOLLOW(B) $= \varnothing$;
  - -Dacă $A \rightarrow \alpha\bullet$, $B \rightarrow \beta \bullet a\gamma \in$ t atunci a $\notin$ FOLLOW(A).

- Analiza sintactică SLR(1) este similară cu cea LR(0);tabela de analiză sintactică are două componente:
  - -Prima, numită ACŢIUNE, determină dacă parserul va face deplasare respectiv reducere, în funcţie de starea ce se află în topul stivei şi de simbolul următor din intrare
  - -Cea de a doua, numită GOTO, determină starea ce se va adăuga în stivă în urma unei reduceri.

Construcţia tabelei de parsare SLR(1)

- Intrare:
  - Gramatica G = (N, T, S, P) augmentată cu S' → S;
  - Automatul M = (Q, Σ, g, t_0, Q);
  - Mulţimile FOLLOW(A), A∈V
- Ieşire:
  - Tabela de analiză SLR(1) compusă din două părţi:
  - ACŢIUNE(t, a), t ∈ Q, a ∈ T ∪ { # },
  - GOTO(t, A), t ∈ Q, A ∈ N.

Construcţia tabelei de parsare SLR(1)

- for(t ∈ Q)
  - for (a ∈ T) ACTIUNE(t, a) = “eroare”;
  - for (A ∈ V) GOTO(t, A) = “eroare”;
- for(t ∈ Q}{
  - for(A→ α•aβ ∈ t)
    - ACTIUNE(t,a)=“D g(t, a)”;//deplasare in g(t, a)
  - for(B→ γ• ∈ t){// acceptare sau reducere
    - if(B == ‘S’) ACTIUNE(t, a) = “acceptare”;
    - else
      - for(a ∈ FOLLOW(B)) ACTIUNE(t,a)=“R B→γ”;
  - }// endfor
  - for (A ∈ N) GOTO(t, A) = g(t, A);
- }//endfor

Parsarea SLR(1)

- **Deplasare**: $(\sigma t, au\#, \pi) \vdash (\sigma tt', u\#, \pi)$ dacă ACTIUNE(t, a)=Dt';
- **Reducere**: $(\sigma t\sigma' t', u\#, \pi) \vdash (\sigma tt'', u\#, \pi r)$ ACTIUNE(t, a) = Rp unde p= $A \rightarrow \beta$, $|\sigma' t'| = |\beta|$ şi t''= GOTO(t, A);
- **Acceptare**: $(t_0 t, \#, \pi)$ dacă ACTIUNE(t,a) = "acceptare"; Analizorul se opreşte cu acceptarea cuvântului de analizat iar π este parsarea acestuia (şirul de reguli care s-a aplicat, în ordine inversă, în derivarea extrem dreaptă a lui w).
- **Eroare**: $(\sigma t, au\#, \pi) \vdash$ eroare dacă ACTIUNE(t,a) = "eroare"; Analizorul se opreşte cu respingerea cuvântului de analizat.

Parsarea SLR(1)

- Intrare:
  - Gramatica G = (N, T, S, P) care este SLR(1) ;
  - Tabela de parsare SLR(1) (ACTIUNE, GOTO);
  - Cuvântul de intrare w $\in$ T*.
- Ieşire:
  - Analiza sintactică (parsarea) ascendentă a lui w dacă w $\in$ L(G);
  - eroare, în caz contrar.
- Se foloseşte stiva St pentru a implementa tranziţiile deplasare/reducere

Parsarea SLR(1)

- char ps[] = “w#”; //ps este cuvantul de intrare w
- int i = 0; // pozitia curenta in cuvantul de intrare
- St.push(t0); // se initializeaza stiva cu t0
- while(true) { // se repeta pana la succes sau eroare
  - t = St.top();
  - a = ps[i] // a este simbolul curent din intrare
  - if(ACTIUNE(t,a) == “acceptare”) exit(“acceptare”);
  - if(ACTIUNE(t,a) == “Dt‘”){
    - St.push(t‘);
    - i++; // se inainteaza in w
  - }//endif
  - else {
    - if(ACTIUNE(t,a) == “R A → $X_1X_2…X_m$”){
      - for(i = 1; i ≤m; i++) St.pop();
      - St.push(GOTO(St.top, A));
    - } //endif
    - else exit(“eroare”);
  - }//endelse
- }//endwhile

Exemplu

- 0.S → E, 1.E → E+T, 2.E → T, 3.T → T*F, 4.T → F, 5.F →(E), 6.F → a

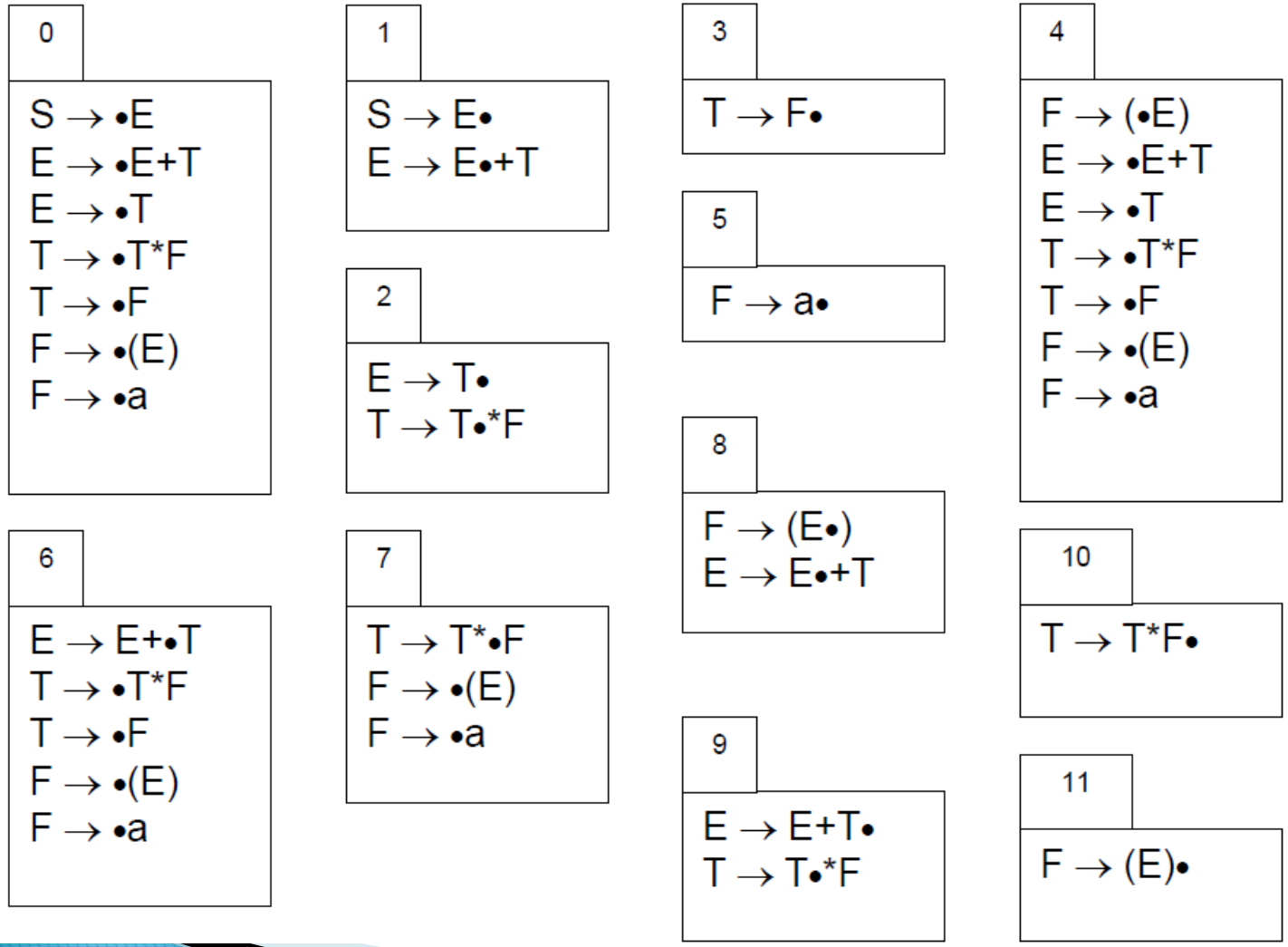


Tabela de tranziţie a automatului LR(0)

g	a	+	*	(	)	E	T	F
0	5			4		1	2	3
1		6						
2			7					
3								
4	5			4		8	2	3
5								
6	5			4			9	3
7	5			4				10
8					11			
9			7					
10								
11								

Tabela de analiză SLR(1)

STARE	ACȚIUNE						GOTO		
	a	+	*	(	)	#	E	T	F
0	D5			D4			1	2	3
1		D6				accepta			
2		R2	D7		R2	R2			
3		R4	R4		R4	R4			
4	D5			D4			8	2	3
5		R6	R6		R6	R6			
6	D5			D4				9	3
7	D5			D4					10
8		D6			D11				
9		R1	D7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

Test SLR(1)

- G nu este LR(0) stările 1, 2, 9 conţin conflict de deplasare/reducere
- FOLLOW(S)={#}, FOLLOW(E)={#,+,)}
- Gramatica este SLR(1) pentru că:
  - în starea 1: $+ \notin$ FOLLOW(S);
  - în starea 2: $* \notin$ FOLLOW(E);
  - în starea 9: $* \notin$ FOLLOW(E).

Stiva	Intrare	Actiune	Iesire
0	a*(a+a)#	deplasare	
05	*(a+a)#	reducere	6.F → a
03	*(a+a)#	reducere	4.T → F
02	*(a+a)#	deplasare	
027	(a+a)#	deplasare	
0274	a+a)#	deplasare	
02745	+a)#	reducere	6.F → a
02743	+a)#	reducere	4.T → F
02742	+a)#	reducere	2.E → T
02748	+a)#	deplasare	

Stiva	Intrare	Actiune	Iesire
027486	a)#	deplasare	
0274865	)#	reducere	6.F → a
0274863	)#	reducere	4.T → F
0274869	)#	reducere	1.E → E+T
02748	)#	deplasare	
02748(11)	#	reducere	5.F →(E)
027(10)	#	reducere	3.T → T*F
02	#	reducere	2.E → T
01	#	acceptare	

Gramatici LR(1)

- **Definiţie**
  - Fie G = (V, T, S, P) o gramatică redusă. Un articol LR(1) pentru gramatica G este o pereche (A → α·β, a), unde A → αβ este un articol LR(0), iar a ∈FOLLOW(A) (se pune # în loc de ε).
- **Definiţie**
  - Articolul (A → β1•β2, a) este valid pentru prefixul viabil αβ1dacă are loc derivarea
    - S dr⇒*αAu ⇒ αβ1β2u
    - iar a = 1:u (a = # dacă u = ε).
- **Teorema**
  - O gramatică G = (V, T, S, P) este gramatică LR(1) dacă şi numai dacă oricare ar fi prefixul viabil φ, nu există două articole distincte, valide pentru φ, de forma(A → α•, a), (B →β•γ, b) unde a ∈FIRST(γb).

Gramatici LR(1)

- Nu există conflict deplasare/reducere. Un astfel de conflict înseamnă două articole (A → α•, a) şi (B → β•aβ', b) valide pentru acelaşi prefix.
- Nu există conflict reducere/reducere. Un astfel de conflict înseamnă două articole complete(A → α•, a) şi (B → β•, a) valide pentru acelaşi prefix
- Pentru a verifica dacă o gramatică este LR(1) se construiește automatul LR(1) în mod asemănător ca la LR(0):
  - Automatul are ca stări mulțimi de articole LR(1)
  - Tranzițiile se fac cu simboluri ce apar după punct
  - Închiderea unei mulțimi de articole se bazează pe faptul că dacă articolul (B → β•Aβ', b) este valid pentru un prefix viabil ▪atunci toate articolele de forma (A → •α, a)unde a ∈FIRTS(αa) sunt valide pentru același prefix.

Procedura de închidere LR(1)

- flag= true;
- while(flag) {
  - flag= false;
  - for ((A→ α•Bβ, a) ∈ I) {
    - for B → γ ∈ P)
      - for(b ∈ FIRST(βa)){
        - if((B → •γ , b) ∉ I) {
          - I = I∪{(B → •γ , b)};
          - flag= true;
        - }//endif
      - }//endforb
    - }//endforB
  - }//endforA
- }//endwhile
- return I;

Automatul LR(1)

- t0 = închidere((S' → ●S,#));T={t_0};marcat(t_0)=false;
- while(∃ t∈T&& !marcat(t)){ // marcat(t) = false
  - for(X ∈ Σ) {
  - t' = Φ;
    - for((A→ α●Xβ ,a) ∈ t)
      - t' = t' ∪{(B→ αX●β ,a) | (B B→ α●Xβ,a) ∈ t};
      - if(t' ≠ Φ){
        - t' = închidere(t');
        - if(t'T) {
          - T= T ∪{ t' };
          - marcat(t') = false;
        - }//endif
        - g(t, X) = t';
      - } //endif
  - } //endfor
  - marcat(t) = true;
- } // endwhile

Automatul LR(1)

- **Teorema**
  - Automatul M construit în algoritmul 2 este determinist şi L(M) coincide cu mulţimea prefixelor viabile ale lui G. Mai mult, pentru orice prefix viabil γ, $g(t_0,\gamma)$ reprezintă mulţimea articolelor LR(1) valide pentru γ.
- Automatul LR(1) pentru o gramatică G, se foloseşte pentru a verifica dacă G este LR(1)
  - Conflict reducere/reducere: Dacă în T există o stare ce conţine articole de forma $(A \rightarrow \alpha\bullet, a)$, $(B \rightarrow \beta\bullet, a)$ atunci gramatica nu este LR(1);
  - Conflict deplasare/reducere: Dacă în Texistă o stare ce conţine articole de forma $(A \rightarrow \alpha\bullet, a)$ şi $(B \rightarrow \beta_1\bullet a\beta_2, b)$, atunci G nu este LR(1).
  - O gramatică este LR(1) dacă orice stare t $\in$Teste liberă de conflicte

Exemplu

- S ▪L=R|R, L ▪*R|a, R ▪L

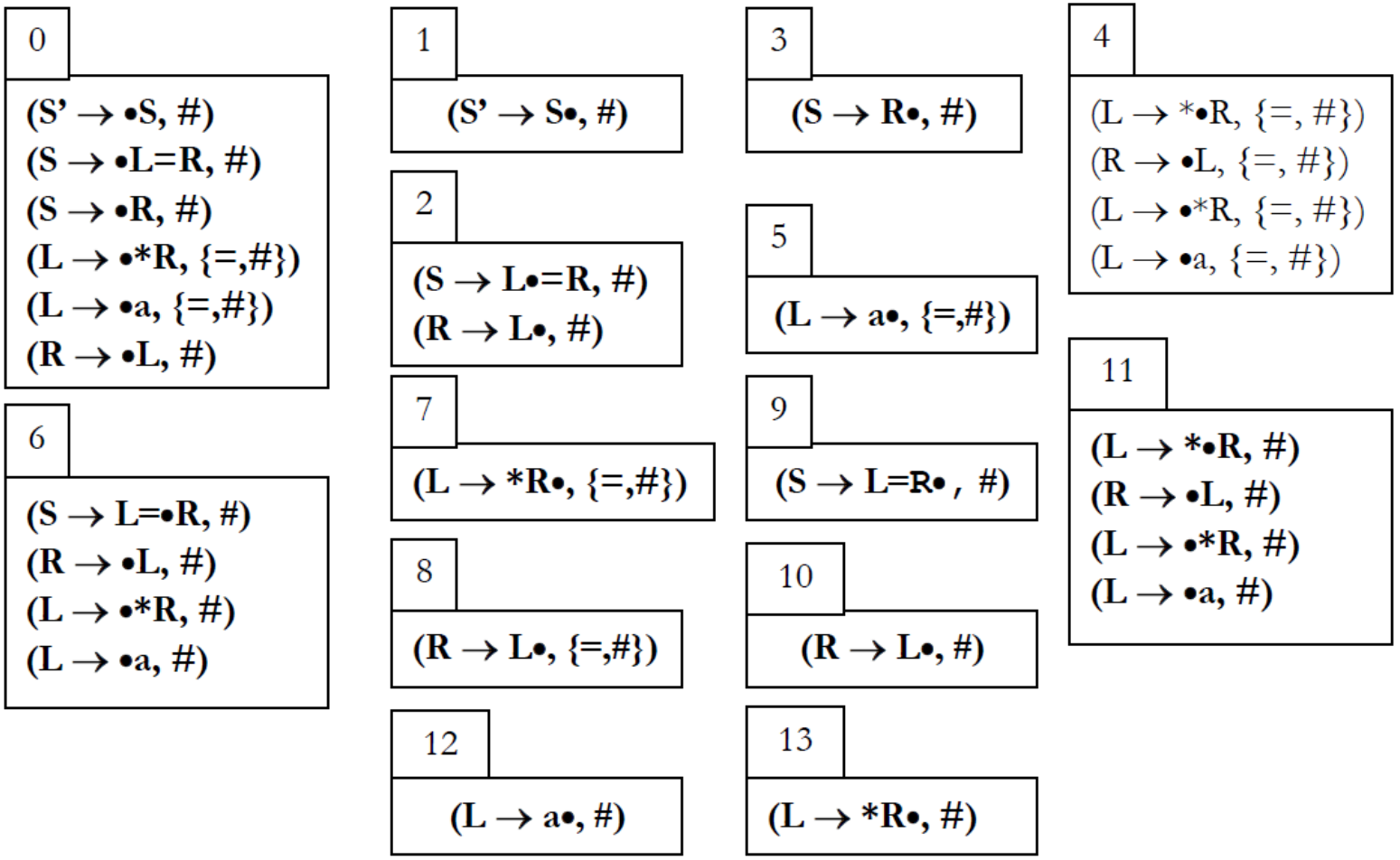


Tabela de tranziţie

g	a	=	*	S	L	R
0	5		4	1	2	3
1						
2		6				
3						
4	5		4		8	7
5						
6	12		11		10	9
7						
8						
9						
10						
11	12		11		10	13
12						
13						

Bibliografie

- Grigoraş Gh., *Construcţia compilatoarelor. Algoritmi fundamentali*, Editura Universităţii "Alexandru Ioan Cuza", Iaşi, 2005