PROGRAMSI PREVODIOCI
- LR gramatike -

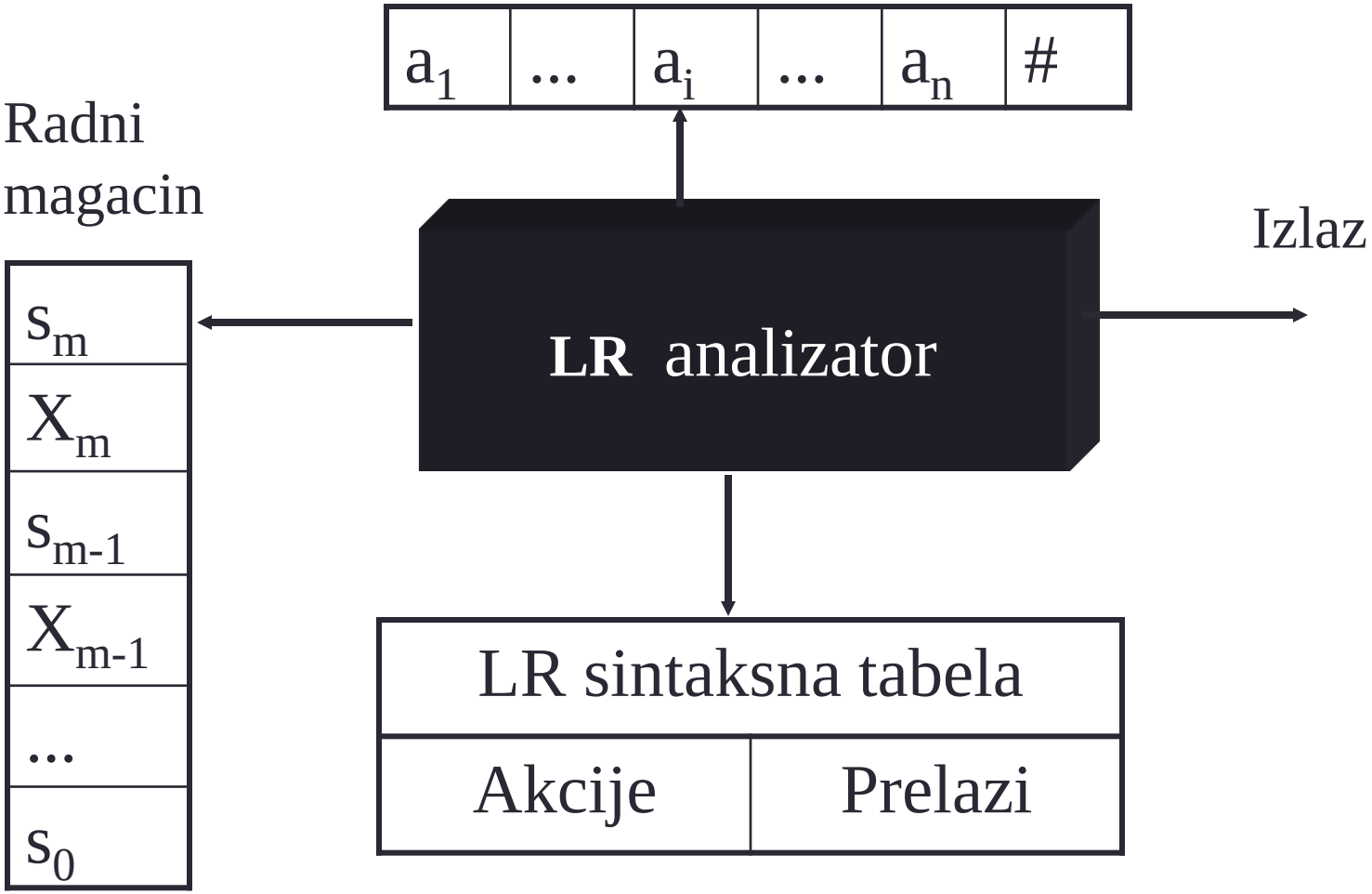
LR(k) analizatori

- L – (Left) Ulazni niz se analizira sleva u desno
- R – (Right) Dobijaju se paravila koja odgovaraju desnom izvođenju
- k – Broj znakova na osnovu kojih se vrši predikcija. Ako *k* nije navedeno podrazumeva se 1 znak.

Osobine LR(k) analizatora

- Mogu da se prepoznaju programske konstrukcije svih jezika koji se mogu opisati beskonteksnim gramatikama.
- LR je najkorišćenij metod za bottom-up analizu bez vraćanja koji se lako implementira.
- Vrlo pogodne za otkrivanje grešaka.
- Generatori sintaksnih analizatora uglavnom koriste ovu metodu za sintaksnu analizu.

Model LR analizatora



LR sintaksna tabela

- Sadrži 2 dela:
  - Akcije - koje se izvršavaju u postupku sintaksne analize zavisno od stanja u kojem se analizator nalazi i tekućeg ulaznog simbola
    - Ovaj deo tabele ima
      - Onoliko vrsta u koliko se stanja analizator može naći
      - Onoliko kolona koliko je terminalnih simbola u gramatici (uključujući i #)
  - Prelaze - koji definišu stanja u koja automat prelazi nakon izvršene redukcije na odredjeni neterminalni simbol
    - Ovaj deo tabele ima
      - Onoliko vrsta u koliko se stanja analizator može naći
      - Onoliko kolona koliko je neterminalnih simbola u gramatici

Akcije u LR sintaksnoj tabeli

- sk (shift k) - znači da u magacin treba smestiti tekući simbol iz ulaznog niza i naredno stanje (k) i preći na analizu sledećeg simbola iz ulaznog niza;
- rk (reduse k) - znači da treba izvršiti redukciju po smeni k (k:$A \rightarrow \beta$) - odnosno, iz magacina treba izbacit 2xlength(β) elemenata, u magacin smestiti neterminalni simbol sa leve strane smene k (A), a tekuće stanje odrediti kao prelaz iz prethodnog stanja u magacinu pod dejstvom tog neterminalnog simbola (A)).
- acc (accept) - znači da je niz prepoznat i dalju analizu treba prekinuti;
- err (error) - znači da u ulaznom nizu postoji greška i teba prekinuti dalju analizu.

Algoritam *shift-reduce* analiza

Postaviti *ip* na početak niza *w#*;
while true begin
Neka je *q* znak u vrhu magacina i *a* znak na koji pokazuje ulazni pokazivač *ip*.
if action(q,a) = sift k **then begin**
Ubaciti u stek *a*, a zatim i *k* i pomeriti ulazni pokazivač *ip* za jedno mesto udesno.
end
else if action(s,a) = reduce k, pri čemu je k-ta smena gramatike A->β **then begin**
Izbaciti **2x|β|** simbola iz steka, ubaciti A u stek, a zatim i oznaka stanja koja se dobija na osnovu goto(s',A), gde je s' oznaka stanja u koje smo se vratili posle redukcije , generiše se izlaz A->β .
end
else if action(s,#) = accept **then**
return
else error()
end

Algoritam

Neka je trenurno stanje analize:

$$(s_0, X_1, s_1, X_2, s_2, \ldots X_m, s_m, \quad a_i, a_{i+1}, \ldots a_n, \#)$$

Ako je $action[s_m, a_i] = shift\ k$, novo stanje analize je :

$$(s_0, X_1, s_1, X_2, s_2, \ldots X_m, s_m, a_i, k, \quad a_{i+1}, \ldots a_n, \#)$$

Ako je $action[s_m, a_i] = reducek$ i $k-$ta smena $A \to \beta$,
novo stanje analize je :

$$(s_0, X_1, s_1, X_2, s_2, \ldots X_{m-r}, s_{m-r}, A, s, \quad a_i, a_{i+1}, \ldots a_n, \#)$$

gde je : $s = goto[s_{m-r}, A], \quad r = |\beta|$

LR sintaksna analiza - Primer

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow \mathbf{id}$

	ACTION						GOTO		
	id	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

LR sintaksna analiza – Primer: id * id

	ACTION						GOTO		
	id	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

STACK	INPUT	ACTION
0	id * id #	s5
0 id 5	* id #	r6 F→id
0 F 3	* id #	r4 T→F
0 T 2	* id #	s7
0 T 2 * 7	id #	s5
0 T 2 * 7 id 5	#	r6 F→id
0 T 2 * 7 F 10	#	r3 T→T*F
0 T 2	#	r2 E→T
0 E 1	#	acc

Vidljivi prefiksi

- U rečeničnoj formi $\phi\beta t$ gde je β fraza koja se redukuje, vidljivi prefiksi su svi prefiksi niza β uključujući i $\phi\beta$.

Za $\phi\beta=u_1u_2...u_r$ prefiksi su $u_1u_2...u_i$ $(1<=i<=r)$

LR(k) gramatike

- Posmatrajmo jedan korak u desnom izvođenju nekog niza: $\phi Bt \rightarrow \phi\beta t$, gde je B prvi neterminal sa desne strane koji se preslikava u niz β primenom pravila $B \rightarrow \beta$.
- (Očigledno je niz t sastavljen od terminalnih simbola)
- Gramatika G je LR(k) gramatika ako se za bilo koji ulazni niz, u svakom koraku izvođenja fraza (podniz) β može detektovati na osnovu prefiksa $\phi\beta$ i skaniranjem najviše k znakova niza t.

Tipovi LR analizatora

Tip odredjuje algoritam za generisanje LR sintaksne tabele:

- SLR – *Simple LR* : Jednostavan za konstrukciju ali postoje gramatike za koje se ne može generisati SLR tabela sintaksne analize.
- Kanonički LR – Najmoćniji ali i najskuplji i najsloženiji postupak projektovanja
- LALR- *Look-a-head Left Right* - Između prva dva i po ceni i po performansama.

SLR analizatori

Osnovna ideja je da se generiše konačan automat koji će da prepoznaje sve vidljive prefikse koji mogu da nastanu u toku generisanja reči jezika koji je opisan datom gramatikom.

LR(0) članovi

•LR(0) član izveden iz nekog pravila sadrži tačku na desnoj strani koja razdvaja deo pravila koji pripada vidljivom prefiksu od dela koji još uvek nije prepoznat.

•LR(0) član je pravilo sa umetnutom tačkom na bilo kojoj poziciji na desnoj strani.

- Članovi izvedeni iz smene : $A \rightarrow XZY$

$$A \rightarrow \cdot XZY$$

$$A \rightarrow X \cdot YZ$$

$$A \rightarrow XY \cdot Z$$

$$A \rightarrow XYZ \cdot$$

Kreiranje SLR sintaksne tabele

- *Korak 1:* Kreiranje kanoničkih skupova LR(0) članova
  - Korišćenjem operacija zatvaranja (*closure*) i prelaza (*go to*)
- *Korak 2:* Kreiranje grafa prelaza konačnog automata za prepoznavanje vidljivih prefiksa
- *Korak 3:* Popunjavanje sintaksne tabele

Operacija zatvaranja (*Closure Operation*)

Ako je I skup LR(0) članova gramatike G, tada je closure(I) skup LR(0) članova koji se dobijaju primenom sledećih pravila:

1. Inicijalno svaki LR(0) član skupa I uključuje se u skup closure(I) .
2. Ako $A \rightarrow \alpha.B\beta$ pripada skupu closure(I) i $B \rightarrow \gamma$ je pravilo gramatike, tada se skupu closure(I) pridodaje i član $B \Rightarrow .\gamma$, ako već ne postoji u tom skupu.

GOTO Operacija

Definiše se kao funkcija *goto*(I,X) gde je I skup LR(0) članova koji sadrži bar jedan član oblika $[A \rightarrow \alpha.X\beta]$, a X znak azbuke.

goto(I,X) je zatvaranje svih članova oblika

$[A \rightarrow \alpha X.\beta]$ pri čemu $[A \rightarrow \alpha.X\beta] \in$ I.

Kreiranje kanoničkog skupa LR članova

1. Ako je S startni simbol gramatike, gramatiku dopuniti smenom S'$\rightarrow$S.
2. Naći zatvaranje skupa LR članova I=$\{$(S' $\rightarrow$ .S)$\}$ i to obeležiti kao skup pravila I_0.
3. Naći sva zatvaranja skupova LR članova I_i=goto(I_k,X).

Kreiranje kanoničkog skupa LR članov -Primer

1. $E \to E + T$
2. $E \to T$
3. $T \to T * F$
4. $T \to F$
5. $F \to (E)$
6. $F \to \mathbf{id}$

I_0 :

$E' \to \cdot E$

$E \to \cdot E + T$

$E \to \cdot T$

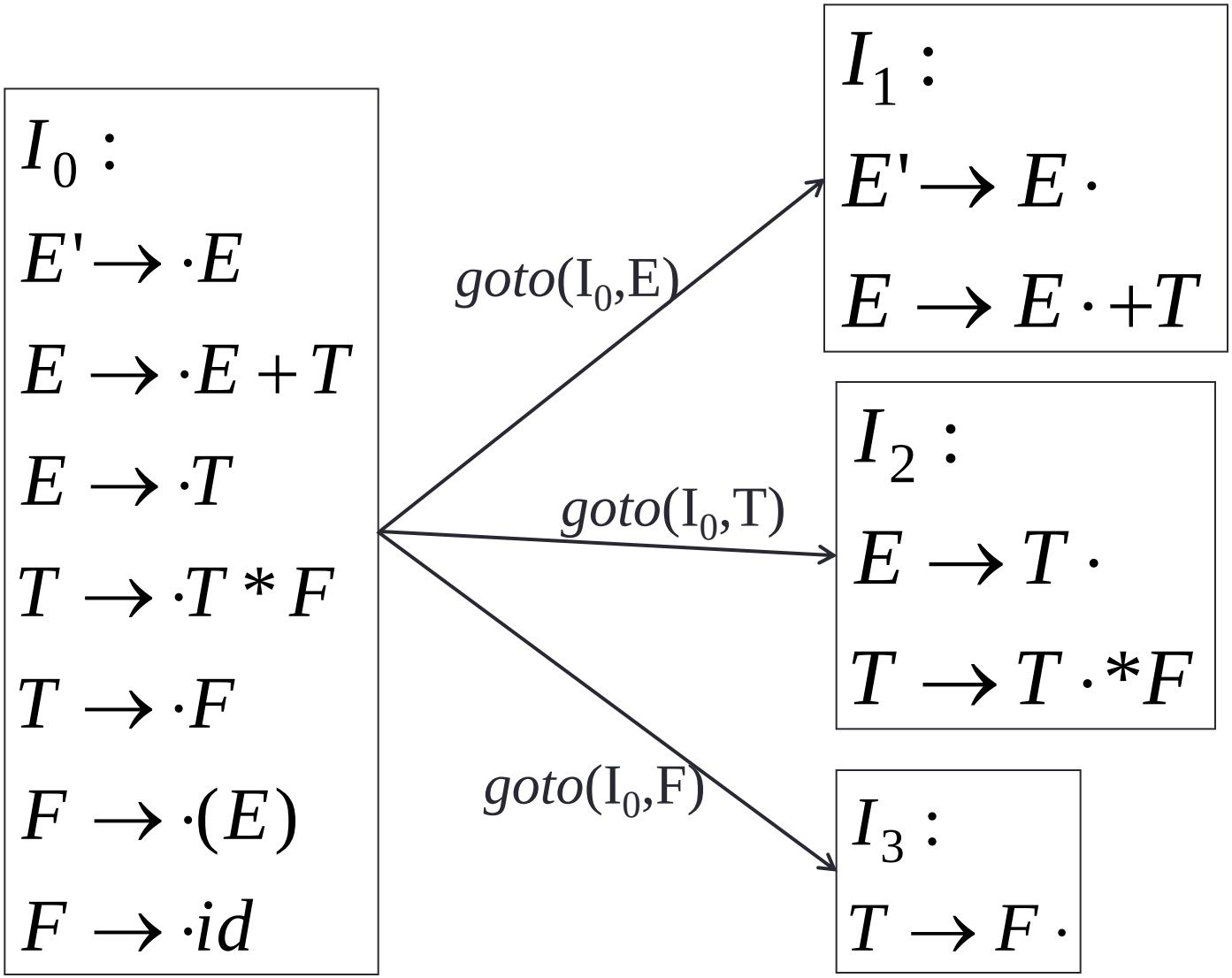
$T \to \cdot T * F$

$T \to \cdot F$

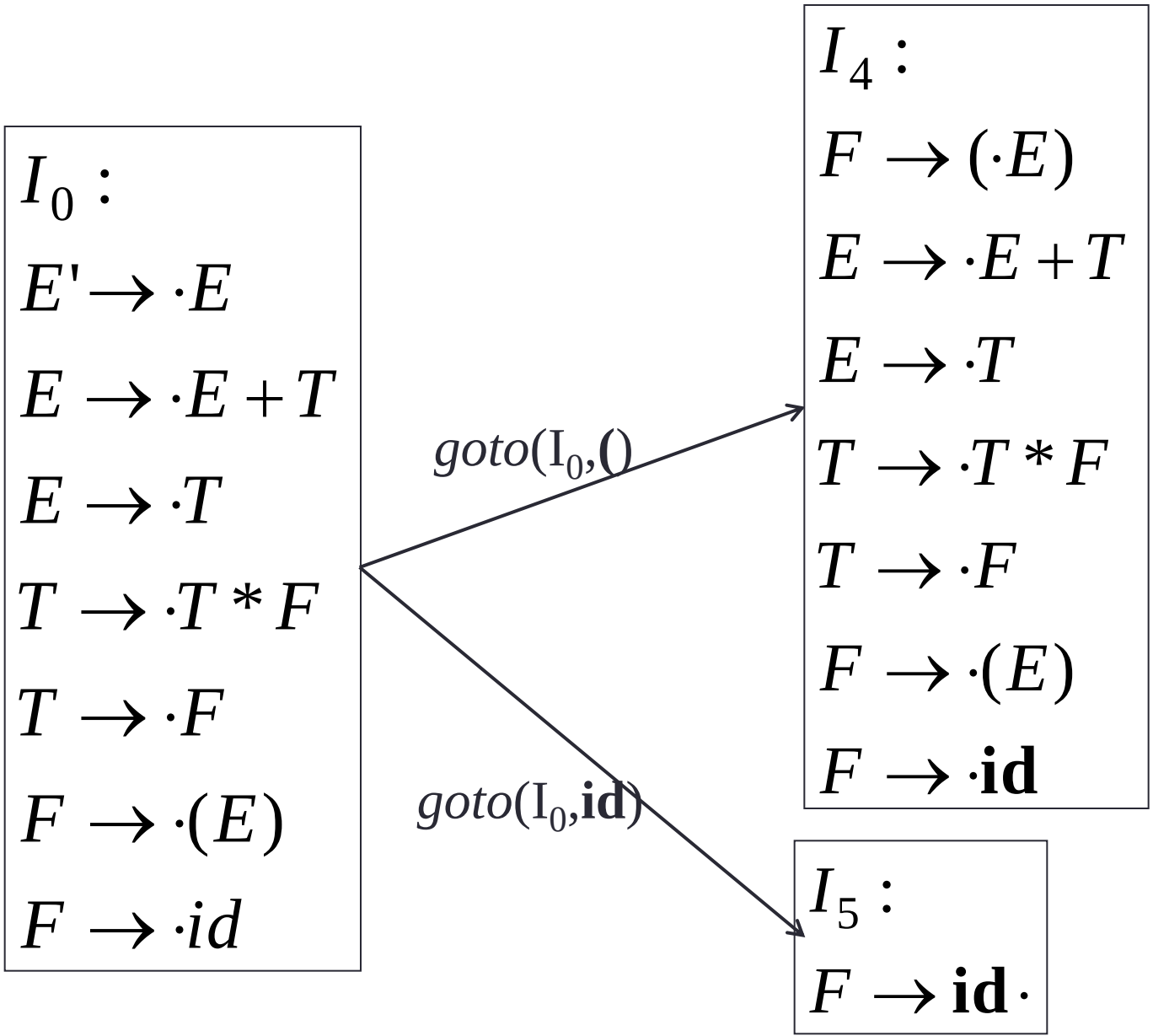
$F \to \cdot (E)$

$F \to \cdot \mathbf{id}$

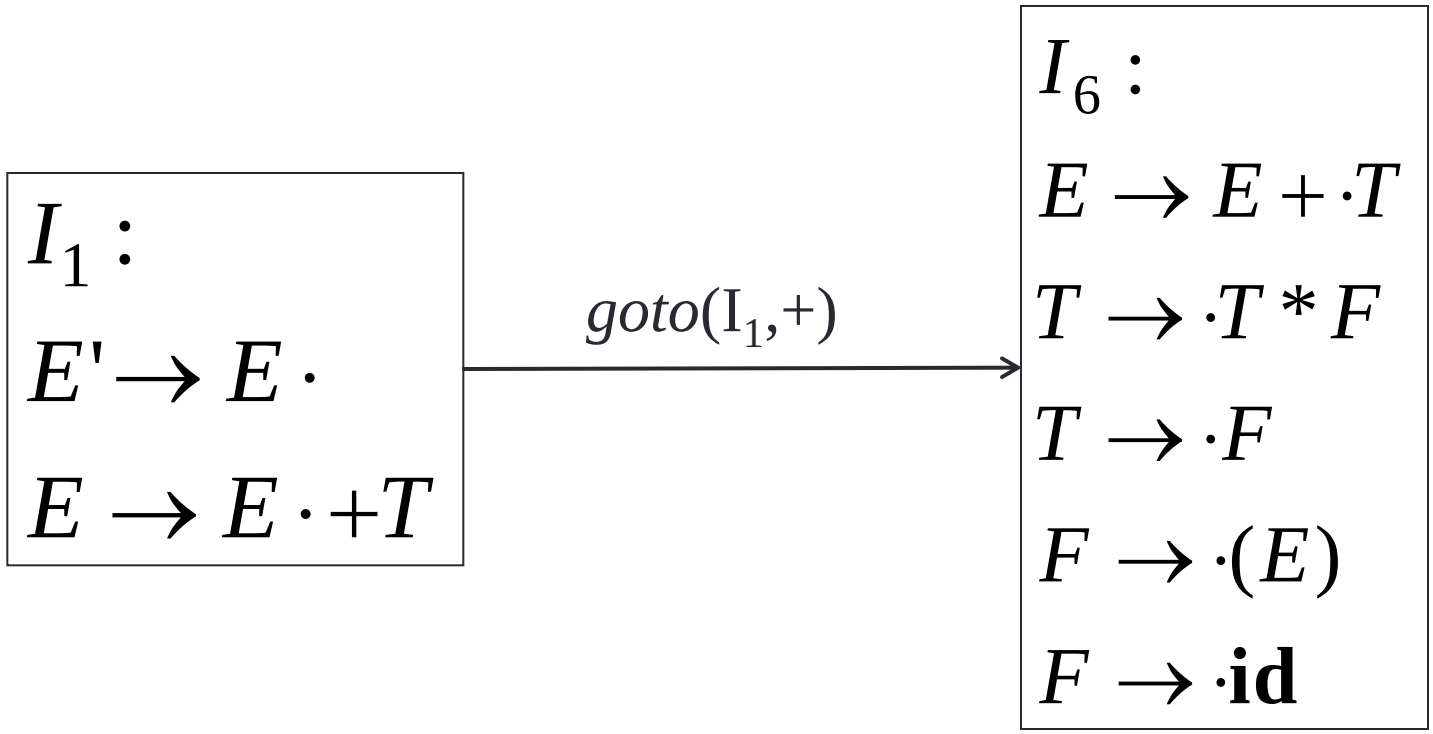
Kreiranje kanoničkog skupa LR članova -Primer



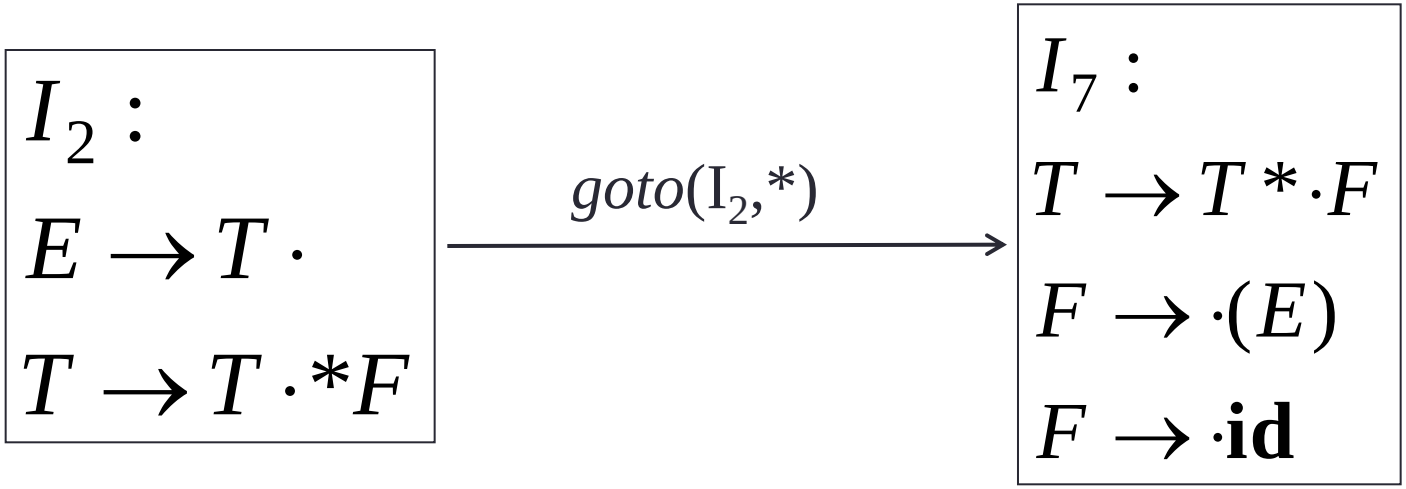
Kreiranje kanoničkog skupa LR članova -Primer



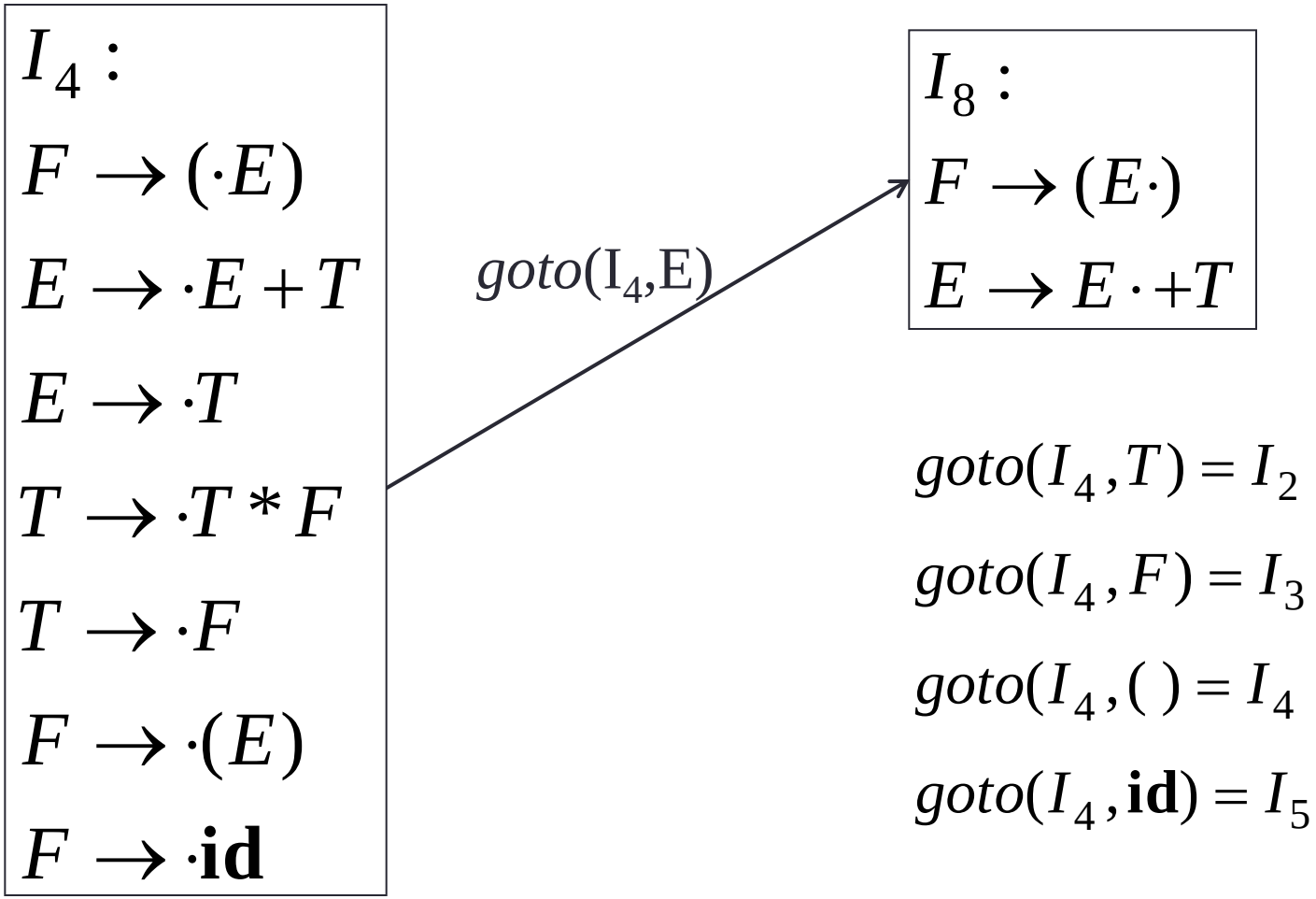
Kreiranje kanoničkog skupa LR članova -Primer



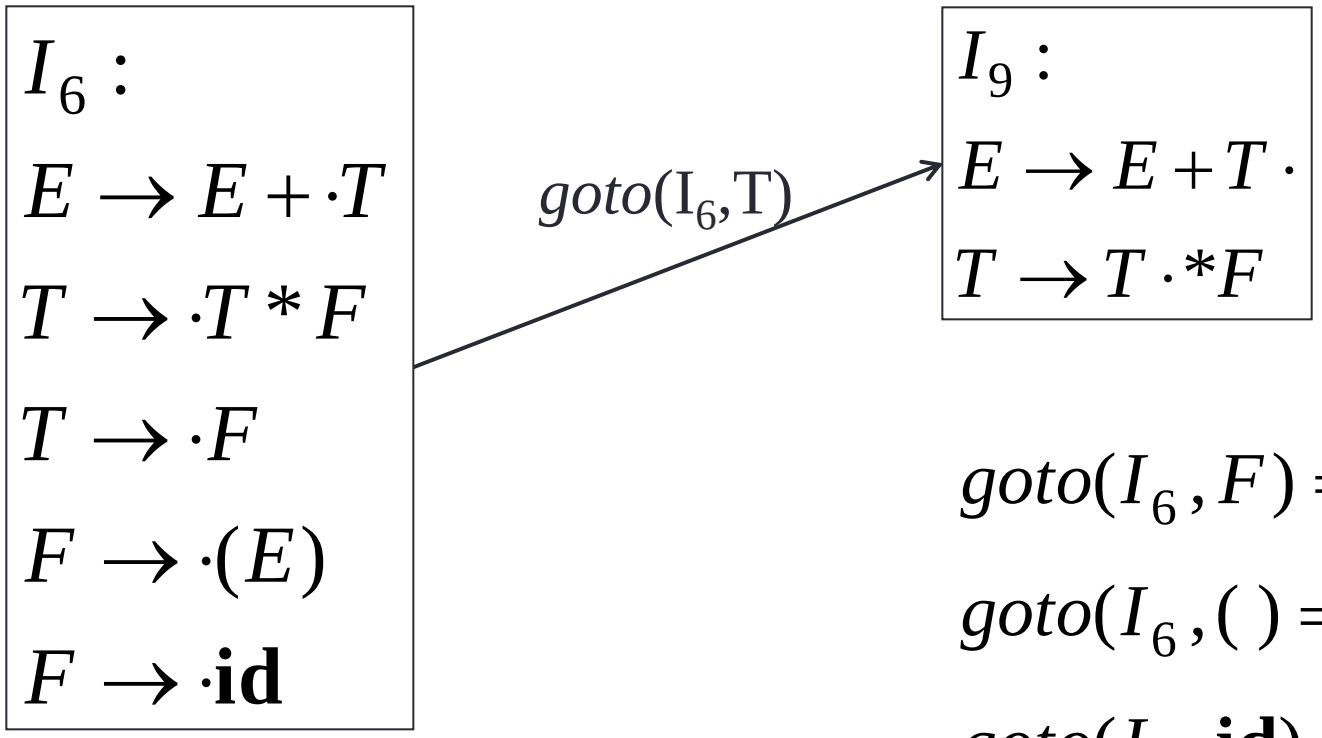
Kreiranje kanoničkog skupa LR članova -Primer



Kreiranje kanoničkog skupa LR članova -Primer



Kreiranje kanoničkog skupa LR članova -Primer

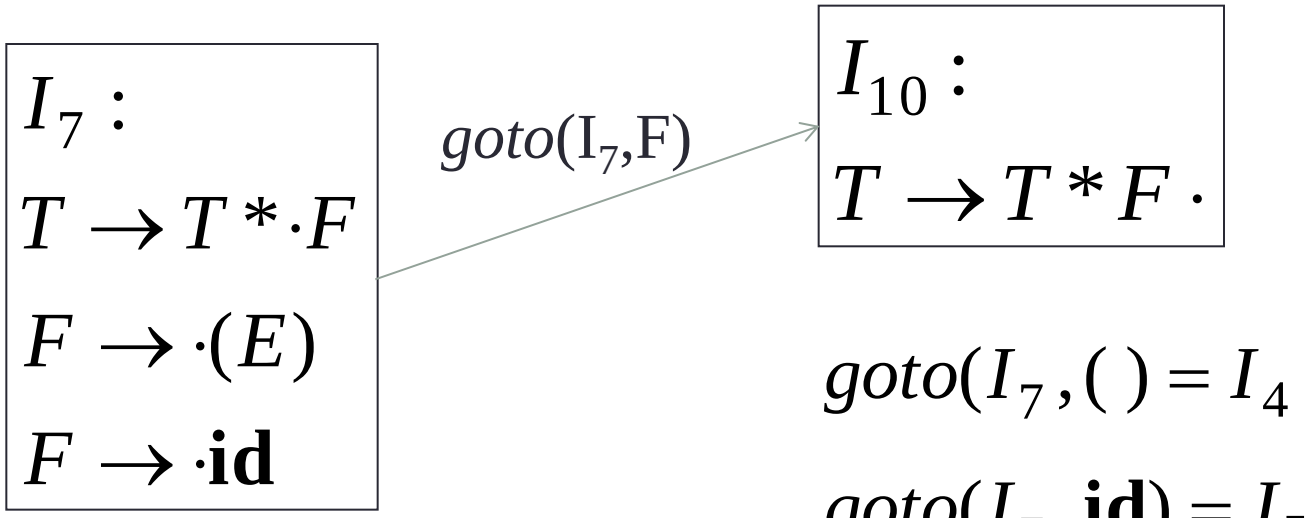


$$goto(I_6, F) = I_3$$

$$goto(I_6, ( ) = I_4$$

$$goto(I_6, \mathbf{id}) = I_5$$

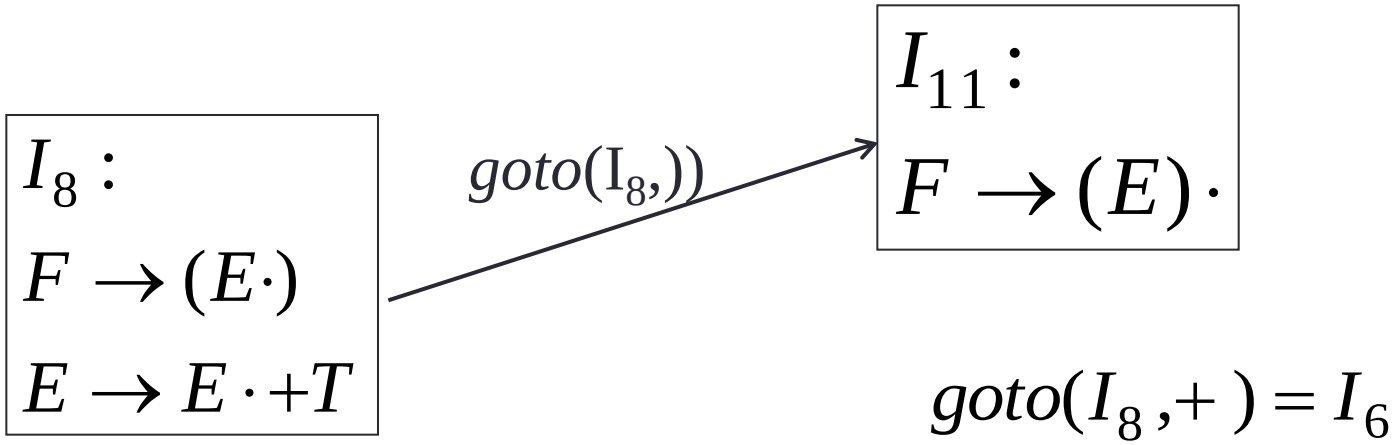
Kreiranje kanoničkog skupa LR članova -Primer



$goto(I_7, ( ) = I_4$

$goto(I_7, \mathbf{id}) = I_5$

Kreiranje kanoničkog skupa LR članova -Primer



$$goto(I_8,) = I_6$$

Kreiranje kanoničkog skupa LR članova -Primer

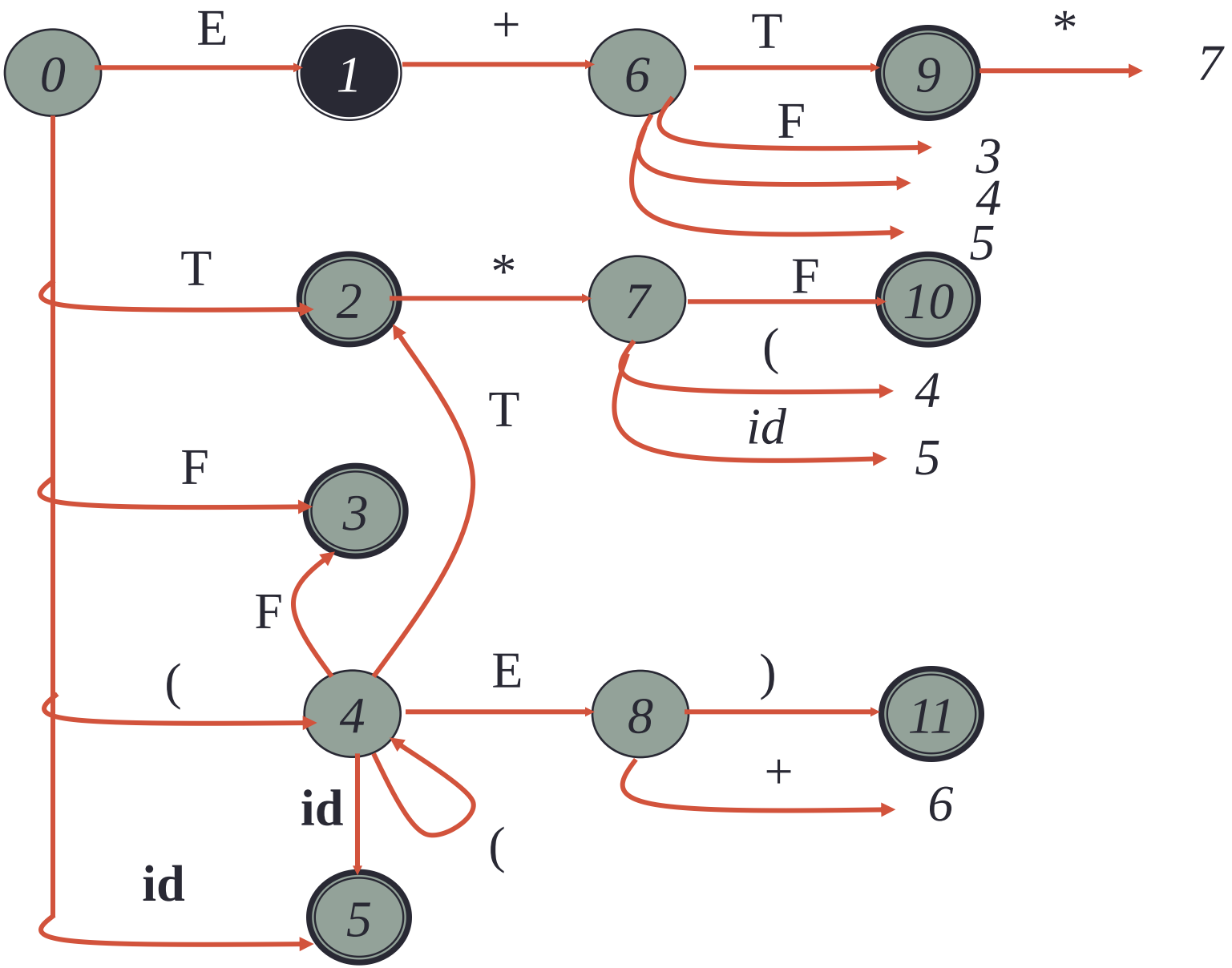
$$I_9:$$
$$E \rightarrow E + T \cdot$$
$$T \rightarrow T \cdot * F$$

$$goto(I_9, *) = I_7$$

Kreiranje grafa prelaza konačnog automata za prepoznavanje vidljivih prefiksa

- Svakom zatvaranju LR pravila iz kanoničkog skupa pravila dodeljuje se jedno stanje u grafu automata.
- Potezi u grafu određeni su goto funkcijama iz kanoničkog skupa LR pravila.
- Ukoliko u nekom zatvaranju postoji pravilo u kojem se . (tačka) nalazi na krajnjoj desnoj poziciji, odgovarajuće stanje obeležiti kao završno. To je redukciono stanje za smenu iz koje je izvedeno dato pravilo.

Graf automata za prepoznavanje vidljivih prefiksa



Popunjavanje LR sintaksne tabele

- Vrste u LR sintaksnoj tabeli su stanja LR sintaksne analize, tj. stanja konačnog automata za prepoznavanje vidljivih prefiksa.
  - Ako u grafu postoji poteg između čvorova *i* i *k* pod dejstvom terminalnog simbola **a**, tada je akcija(*i*,**a**)=s*k*.
  - Ako u grafu postoji poteg između čvorova *i* i *k* pod dejstvom neterminalnog simbola A, tada je prelaz(*i*,A)=k.
  - Ako zatvaranju li pripada član oblika $A \rightarrow \alpha.$ (a smena pod rednim brojem k ima oblik: $A \rightarrow \alpha$), tada će u grafu stanje *i* biti označeno kao završno stanje što će značiti da je to redukciono stanje za smenu *k*. U tom slučaju akcija(*i*,**a**)=r*k* za svako **a** koje pripada skupu FOLLOW(A).

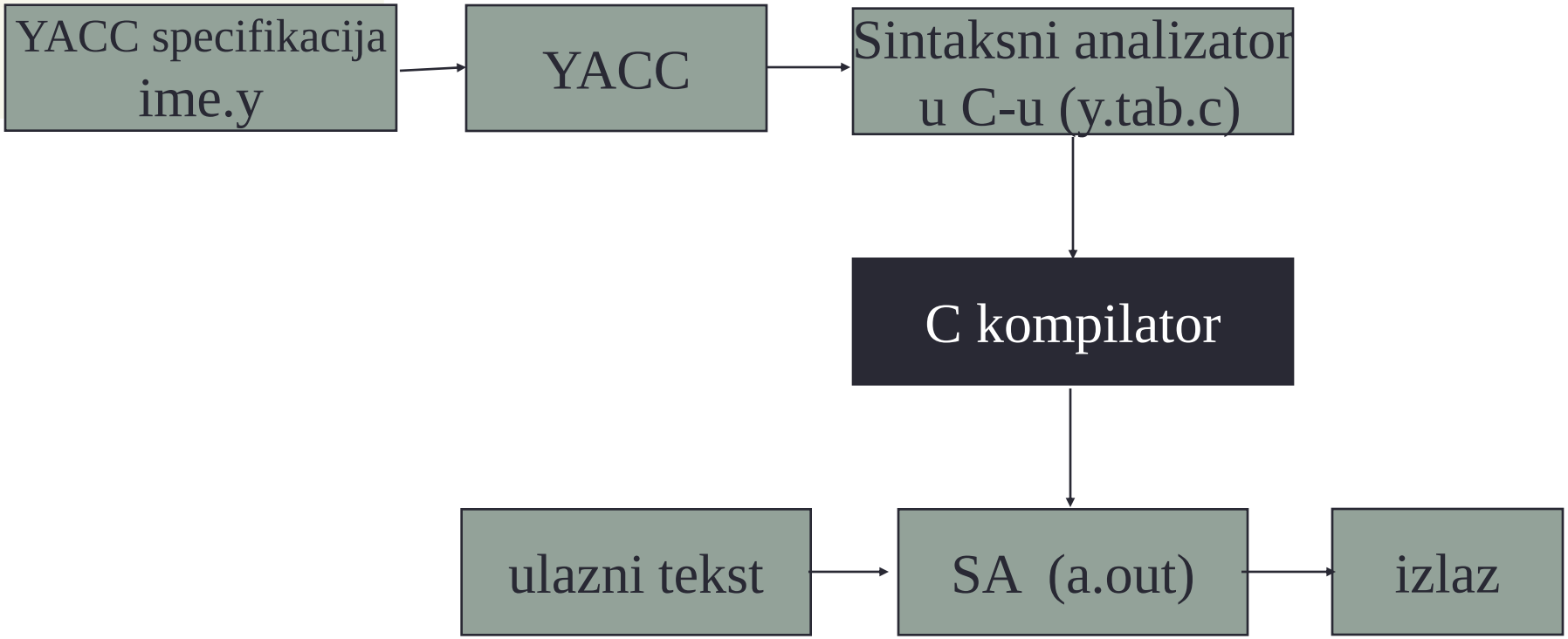
Sintaksna tabela

STATE	ACTION						GOTO		
	id	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Generatori sintaksnih analizatora

- YACC – Nastao 1975. godine kao deo UNIX operativnog sistema. Generiše kod u C-u. Radi u sprezi sa LEX-om.
- BISON – Besplatna verzija YACC-a. Generiše kod u C-u. Radi u sprezi sa FLEX-om.
- CUP – Takodje „Open Source“ alat. Generiše kod u Javi. Radi u spezi sa JFLEX-om.

YACC


YACC specifikacija
ime.y
YACC
Sintaksni analizator
u C-u (y.tab.c)
C kompilator
ulazni tekst
SA (a.out)
izlaz

Struktura YACC specifikacije

definicije
%%
pravila
%%
korisnički potprogrami

Definicije

- Kao i u LEX specifikaciji ovaj deo može da sadrži zaglavlje koje se umeće direktno u C kod koji se generiše.
- Definišu se terminalni simboli gramatike (tokeni), startni simbol, prioritet i asoscijativnost operatora

```
%start <ime_startnog_simbola>
%token<tok1>[<tok2>]…

%left ‘+’,’-’
%left ‘*’,’/’
%right ’STEPEN’
```

Pravila

- Za svaku smenu gramatike kreira se po jedno pravilo:
- Pravilo sadrži smenu i akciju.

$A \rightarrow \beta$ se zapisuje kao: $A : \beta$

- Akcija predstavlja programski kod koji se pridružuje smeni.
- Ovde mogu da budu pozvane i neke funcije koje su definisane u trećem delu.

YACC specifikacija – Primer:

Kreirati YACC specifikaciju za generisanje sintaksnog analizatora za gramatiku:

E=>E+E| E-E| E*E| E/E| (E)| broj

```
%{
#include<ctype.h>
%}
%token BROJ
%left ‘+’ ’-’
%left ‘*’ ‘/’
```

YACC specifikacija – Primer:

```
%%
E1 :  E '\n'    { puts("prepoznat aritmeticki iztraz");};
E      :  E '+' E
       | E '-' E
       | E '*' E
       | E '/' E
       | '(' E ')'
       | BROJ
       ;
```

YACC specifikacija – Primer:

```
%%
int yylex()
{
  int c;
  while(( c=getchar())==‘ ‘);
    if(isdigit( c )) return BROJ;
  return c;
}
```

Leksički analizator implementiran kao sastavni deo YACC specifikacije.