CURSUL 8

PROPRIETĂȚI ALE GRAMATICILOR DE TIP *LR*

Teorema 1. Automatul finit determinist construit pe baza funcției $goto$ din algoritmul $LR(1)$ recunoaște mulțimea prefixelor viabile ale lui G.

Demonstrație. 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 $L(G) = L(G')$. Considerăm automatul $A = (Q, \Sigma \cup N, goto, I_0, F)$, unde $Q = C_G = \{I_0, I_1, \dots, I_n\}$, $I_0 = closure(\{S' \to .S; \#\})$, $goto$ fiind funcția de tranziție a lui A, C_G mulțimile canonice $LR(1)$ asociate lui $G, F = \{J \in C_G | \exists A \to \alpha.; a \in J\}$.

Arătăm că $\forall A \neq S', A \to \alpha.\beta; a \in goto(I_0, \gamma) \Leftrightarrow A \to \alpha.\beta; a$ validă pentru prefixul viabil $\gamma \in (N \cup \Sigma)^*$.

Teorema 2. Gramatica independentă de context $G = (N, \Sigma, S, P)$ este $LR(1)$ dacă și numai dacă tabela $action$ construită cu algoritmul $LR(1)$ nu are intrări multiple.

Teorema 3. Orice gramatică de tip $LR(k), k \geq 2$, este echivalentă cu o gramatică de tip $LR(1)$.

Pentru algoritmii de tip LR este suficient $k = 1$, adică se ia în considerare doar un singur simbol lookahead.

Teorema 4. Orice gramatică de tip $LL(k), k \geq 0$, este echivalentă cu o gramatică de tip $LR(k)$.

GRAMATICI DE TIP *LALR*(1)

Fie $G = (N, \Sigma, S, P)$ gramatică independentă de context de tip $LR(1)$, extensia sa $G' = (N \cup \{S'\}, \Sigma \cup \{\#\}, S', P \cup \{S' \to S\})$ cu $L(G) = L(G')$. Fie $C_G = \{I_0, I_1, \dots, I_n\}$ mulțimile canonice $LR(1)$ asociate, $I_0 = closure(\{S' \to .S; \#\})$.

Definiție. Definim nucleul unei mulțimi de configurații din C_G astfel:

- $S' \rightarrow .S$ este nucleul lui I_0
- Fie $I_j \in C_G, X \in N \cup \Sigma$ astfel încât $goto(I_j, X) \neq \emptyset$. Atunci
$$K = \{A \rightarrow \alpha X.\beta \mid \exists A \rightarrow \alpha.X\beta; a \in I_j\}$$
este nucleul multimii canonice $goto(I_j, X)$.

Definiție. Fie $G = (N, \Sigma, S, P)$ gramatică $LR(1)$, $C_G = \{I_0, I_1, \dots, I_n\}$, mulțimile canonice $LR(1)$ asociate. Partiționăm C_G în mulțimi disjuncte două câte două, $G_G = S_1 \cup \dots \cup S_m$, unde pentru orice $i, 1 \leq i \leq m, S_i$ conține toate mulțimile canonice $LR(1)$ din C_G care au același nucleu, K_i. Evident, $S_i \cap S_j = \emptyset, K_i \neq K_j, \forall i \neq j, 1 \leq i, j \leq m$.
Fie S_i o astfel de mulțime, $S_i = \{J_1, \dots, J_{s_i}\} \subseteq C_G$, unde $J_1, \dots, J_{s_i}$ au același nucleu, K_i. Pentru fiecare configurație $A \rightarrow \alpha.\beta \in K_i$ vom reuni simbolurile lookahead ale configurațiilor $LR(1)$ corespondente din fiecare mulțime $J_1, \dots, J_{s_i}$, apoi aplicăm aceeași unificare pentru toate celelalte configurații care nu sunt în nucleu și care au fost introduse în fiecare dintre mulțimile $J_1, \dots, J_{s_i}$ prin operația $closure$, plecând de la configurații din nucleul K_i.

Se obține astfel o nouă mulțime de configurații $LR(1)$, notată cu Z_i, având nucleul K_i și care va înlocui mulțimile $J_1, \dots, J_{s_i}$.

De asemenea, tranzițiile dintre mulțimile noi $Z_1, \dots Z_m$, date de $goto$, vor fi consistente cu tranzițiile inițiale dintre mulțimile $I_0, I_1, \dots, I_n$.

Dacă prin această operație de unificare în urma căreia se obțin mulțimile $Z_1, \dots, Z_m$ nu rezultă conflicte în tabela $action$, atunci spunem că gramatica G este de tip $LALR(1)$ (LA provine de la *Look Ahead*).

Observații.

- Aceasta este o metodă folosită de majoritatea parser-elor de tip LR pentru a reduce din numărul de intrări în tabela de analiză sintactică ($action, goto$).
- Programul $bison$, care este un generator de analizoare sintactice, primește la intrare o gramatică independentă de context, folosind o anumită specificație, și produce la ieșire o tabelă $LALR(1)$.
- Prin utilizarea tabelelor $LALR(1)$ numărul de intrări în tabela de analiză sintactică poate fi redus cu până la 30%.

Exemplu. Se dă gramatica G cu producțiile:

$1: S \rightarrow AA \quad 2: A \rightarrow aA \quad 3: A \rightarrow b$

Adăugăm producția $S' \rightarrow S$ și obținem mulțimile $LR(1)$

$$I_0 = \begin{bmatrix} S' \to .S; \# & \xrightarrow{goto(I_0,S)} I_1 \\ S \to .AA; \# & \xrightarrow{goto(I_0,A)} I_2 \\ A \to .aA; a|b & \xrightarrow{goto(I_0,a)} I_3 \\ A \to .b; \to a|b & \xrightarrow{goto(I_0,b)} I_4 \end{bmatrix}$$

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

$$I_2 = \begin{bmatrix} S \to A.A; \# & \xrightarrow{goto(I_2,A)} I_5 \\ A \to .aA; \# & \xrightarrow{goto(I_2,a)} I_6 \\ A \to .b; \# & \xrightarrow{goto(I_2,b)} I_7 \end{bmatrix}$$

$$I_3 = \begin{bmatrix} A \to a.A; a|b & \xrightarrow{goto(I_3,A)} I_8 \\ A \to .aA; a|b & \xrightarrow{goto(I_3,a)} I_3 \\ A \to .b; a|b & \xrightarrow{goto(I_3,b)} I_4 \end{bmatrix}$$

$$I_4 = [A \to b.; a|b]$$

$$I_5 = [S \to AA.; \#]$$

$$I_6 = \begin{bmatrix} A \to a.A; \# & \xrightarrow{goto(I_6,A)} I_9 \\ A \to .aA; \# & \xrightarrow{goto(I_6,a)} I_6 \\ A \to .b; \# & \xrightarrow{goto(I_6,b)} I_7 \end{bmatrix}$$

$$I_7 = [A \to b.; \#]$$

$$I_8 = [A \to aA.; a|b]$$

$$I_9 = [A \to aA.; \#]$$

Observăm că I_3 și I_6, respectiv I_4, I_7 și I_8, I_9 au același nucleu, deci pot fi unificate.
Se obțin astfel:

$$I_{36} = \begin{bmatrix} A \to a.A; a|b|\# & \xrightarrow{goto(I_{36},A)} I_{89} \\ A \to .aA; a|b|\# & \xrightarrow{goto(I_{36},a)} I_{36} \\ A \to .b; a|b|\# & \xrightarrow{goto(I_{36},b)} I_{47} \end{bmatrix}$$

$$I_{47} = [A \to b.; a|b|\#] \qquad I_{89} = [A \to aA.; a|b|\#]$$

Obținem tabela de simboluri:

	a	b	#	S	A
0	$shift$ 36	$shift$ 47	$error$	1	2
1	$error$	$error$	$accept$	$error$	$error$
2	$shift$ 36	$shift$ 47	$error$	$error$	5
36	$shift$ 36	$shift$ 47	$error$	$error$	89
47	$reduce$ 3	$reduce$ 3	$reduce$ 3	$error$	$error$
5	$error$	$error$	$reduce$ 1	$error$	$error$
89	$reduce$ 2	$reduce$ 2	$reduce$ 2	$error$	$error$

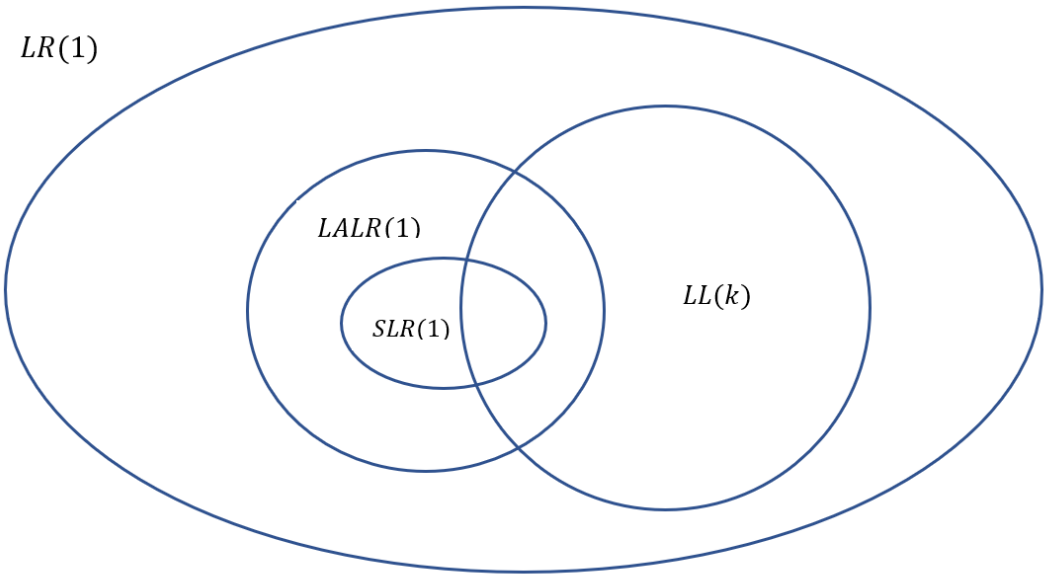
Tabela nou obținută nu are intrări multiple, rezultă că G este $LALR(1)$.

Există un mod mai eficient de a calcula tabelele $LALR(1)$

- Ori de câte ori este creată o nouă stare (mulțime canonică) cu algoritmul $LR(1)$, se verifică imediat dacă se poate face operația de unificare.
- De obicei, operația de unificare presupune și propagarea simbolurilor $lookahead$ adăugate la starea nou construită la pasul de unificare.
  Astfel, în exemplul anterior, în momentul când ar trebui construită I_6, aceasta are același nucleu cu I_3, deci cele două stări se reunesc, obținându-se I_{36}. Din I_3 cu b se ajunge în I_4, care era deja construită. Nu mai are sens să construim explicit I_7 (starea care s-ar fi obținut din I_6 trecând peste b, ci noile simboluri $lookahead$ aduse de I_6 (adică \$), vor fi propagate la I_4.

- Dacă mai este nevoie, această propagare se aplică în continuare.

Teoremă. Există următoarele incluziuni (stricte) între familiile de limbaje $LL(k), SLR(1), LALR(1), LR(1)$:



Anexa 1: Rezolvarea conflictelor în cazul gramaticii ambigue pentru generarea expresiilor

Fie G gramatica cu producțiile:

1: $E \rightarrow E + E$
2: $E \rightarrow E * E$
3: $E \rightarrow n$

Gramatica de mai sus este ambiguă deoarece, de exemplu, șirul $n + n + n$ poate fi derivat cu două derivări stângi distincte.

1) Extindem G la G': introducem productia $E' \rightarrow E$
2) Calculăm mulțimile $Follow$ cu algoritmul din cursul 4, inițializând $Follow(E)$ cu $\{\#\}$. Obținem:

$Follow(X)$	Pasul 1	Pasul 2
E	$\#, +, *$	

3) Calculam mulțimile canonice $LR(0)$ pentru G:

$$I_0 = \begin{bmatrix} E' \to .E \xrightarrow{goto(I_0,E)} I_1 \\ E \to .E + E \xrightarrow{goto(I_0,E)} I_1 \\ E \to .E * E \xrightarrow{goto(I_0,E)} I_1 \\ E \to .n \xrightarrow{goto(I_0,n)} I_2 \end{bmatrix}$$

$$I_1 = \begin{bmatrix} E' \to E. \\ E \to E. + E \xrightarrow{goto(I_1,+)} I_3 \\ E \to E. * E \xrightarrow{goto(I_1,*)} I_4 \end{bmatrix}$$

$$I_2 = [E \to n.]$$

$$I_3 = \begin{bmatrix} E \to E + .E \xrightarrow{goto(I_3,E)} I_5 \\ E \to .E + E \xrightarrow{goto(I_3,E)} I_5 \\ E \to .E * E \xrightarrow{goto(I_3,n)} I_5 \\ E \to .n \xrightarrow{goto(I_3,n)} I_2 \end{bmatrix}$$

$$I_4 = \begin{bmatrix} E \to E * .E \xrightarrow{goto(I_4,E)} I_6 \\ E \to .E + E \xrightarrow{goto(I_4,E)} I_6 \\ E \to .E * E \xrightarrow{goto(I_4,n)} I_6 \\ E \to .n \xrightarrow{goto(I_3,n)} I_2 \end{bmatrix}$$

$$I_5 = \begin{bmatrix} E \to E + E. \\ E \to E. + E \xrightarrow{goto(I_5,+)} I_3 \\ E \to E. * E \xrightarrow{goto(I_5,*)} I_4 \end{bmatrix}$$

$$I_6 = \begin{bmatrix} E \to E * E. \\ E \to E. + E \xrightarrow{goto(I_6,+)} I_3 \\ E \to E. * E \xrightarrow{goto(I_6,*)} I_4 \end{bmatrix}$$

Tabela de analiză sintactică $SLR(1)$ pentru G:

	$action$				$goto$
	$+$	$*$	n	#	E
0	$Error$	$Error$	$Shift\ 2$	$Error$	1
1	$Shift\ 3$	$Shift\ 4$	$Error$	$accept$	$Error$
2	$Reduce\ 3$	$Reduce\ 3$	$Error$	$Reduce\ 3$	$Error$
3	$Error$	$Error$	$Shift\ 2$	$Error$	5
4	$Error$	$Error$	$Shift\ 2$	$Error$	6
5	$Reduce\ 1$ ~~$Shift\ 3$~~	~~$Reduce\ 1$~~ $Shift\ 4$	$Error$	$Reduce\ 1$	$Error$
6	$Reduce\ 2$ ~~$Shift\ 3$~~	$Reduce\ 2$ ~~$Shift\ 4$~~	$Error$	$Reduce\ 2$	$Error$

Tabela *action* are intrări multiple, rezultă că G nu este $SLR(1)$.
Să considerăm că operatorii +, * sunt asociativi la stânga iar * este prioritar față de +.
Vom analiza cele 4 conflicte obținute:

- Conflict $reduce\ 1\ /\ shift\ 3$ pentru '+' în starea 5. Avem de ales între a reduce din stiva de lucru pe $E + E$ la E sau a deplasa pe stiva de lucru operatorul +. Deoarece + este asociativ la stânga, vom face mai întâi reducerea (echivalent cu efectuarea mai întâi a adunării între primul și al doilea termen).
- Conflict $reduce\ 1\ /\ shift\ 4$ pentru '*' în starea 5. Avem de ales între a reduce din stiva de lucru pe $E + E$ la E sau a deplasa pe stiva de lucru operatorul *. Deoarece * este prioritar față de +, vom face mai întâi deplasarea lui * pe stiva de lucru, pentru a putea efectua mai întâi înmulțirea și apoi adunarea.
- Conflict $reduce\ 2\ /\ shift\ 3$ pentru '+' în starea 6. Avem de ales între a reduce din stiva de lucru pe $E * E$ la E sau a deplasa pe stiva de lucru operatorul +. Deoarece * este prioritar față de +, vom face mai întâi reducerea (echivalent cu efectuarea mai întâi a înmulțirii) și apoi deplasarea lui +.
- Conflict $reduce\ 2\ /\ shift\ 4$ pentru '*' în starea 6. Avem de ales între a reduce din stiva de lucru pe $E * E$ la E sau a deplasa pe stiva de lucru operatorul *. Deoarece * este asociativ la stânga, vom face mai întâi reducerea.

În tabelă a fost tăiată acțiunea care nu se efectuează.
În felul acesta se poate proceda cu toți operatorii binari și astfel se simplifică scrierea regulilor sintactice pentru expresii. Simultan, se reduce și numărul de intrări în tabelă.

Anexa 2: Revenirea din eroare într-un parser de tip LR

Pentru fiecare situație de eroare se implementează o rutină de eroare.
Există două modalități:
1) Se analizează starea curentă pentru care s-a obținut eroare. Dacă pentru respectiva stare există un token pentru care avem deplasare sau reducere, atunci se consideră că în intrare ar exista acel token și se efectuează acțiunea corespondentă, de regulă deplasare (adică se plaseaza pe stiva de lucru acel token și noua stare a parser-ului), după care se continuă parsarea. Eventual, putem repeta aceasta dacă în noua stare se obține din nou eroare pentru token-ul curent. Există un număr finit de astfel de repetări posibile.
2) Se ignoră token-ul curent și pentru starea curentă se verifică dacă se poate continua parsarea fără eroare pentru următorul token. Repetând acest pas, în caz că se obține în continuare eroare, se va afișa întregul șir de erori, fie până la întalnirea unei configurații din care se poate continua fără eroare, fie până la finalul fișierului.

Considerăm gramatica G din Anexa 1, cu producțiile:

1: $E \rightarrow E + E$
2: $E \rightarrow E * E$
3: $E \rightarrow n$

Tabela $SLR(1)$, după eliminarea ambiguităților (conflictelor), este:

	$+$	$*$	n	#	E
0	$Error1()$	$Error1()$	$Shift\ 2$	$Error2()$	1
1	$Shift\ 3$	$Shift\ 4$	$Error3()$	$accept$	
2	$Reduce\ 3$	$Reduce\ 3$	$Error3()$	$Reduce\ 3$	
3	$Error1()$	$Error1()$	$Shift\ 2$	$Error1()$	5
4	$Error1()$	$Error1()$	$Shift\ 2$	$Error1()$	6
5	$Reduce\ 1$	$Shift\ 4$	$Error3()$	$Reduce\ 1$	
6	$Reduce\ 2$	$Reduce\ 2$	$Error3()$	$Reduce\ 2$	

unde pentru fiecare situație de eroare s-a implementat o rutină de eroare:
$Error1() = \{ print(\text{“}se\ asteapta\ un\ operand\text{”}); push\ \text{‘}n\text{’}; push\ 2; \}$
$Error2() = \{print(\text{“}Expresie\ vida\text{”}); \}$
$Error3() = \{print(\text{“}Se\ asteapta\ un\ operator\text{”}); push\ \text{‘}+\text{’}; push\ 3; \}$