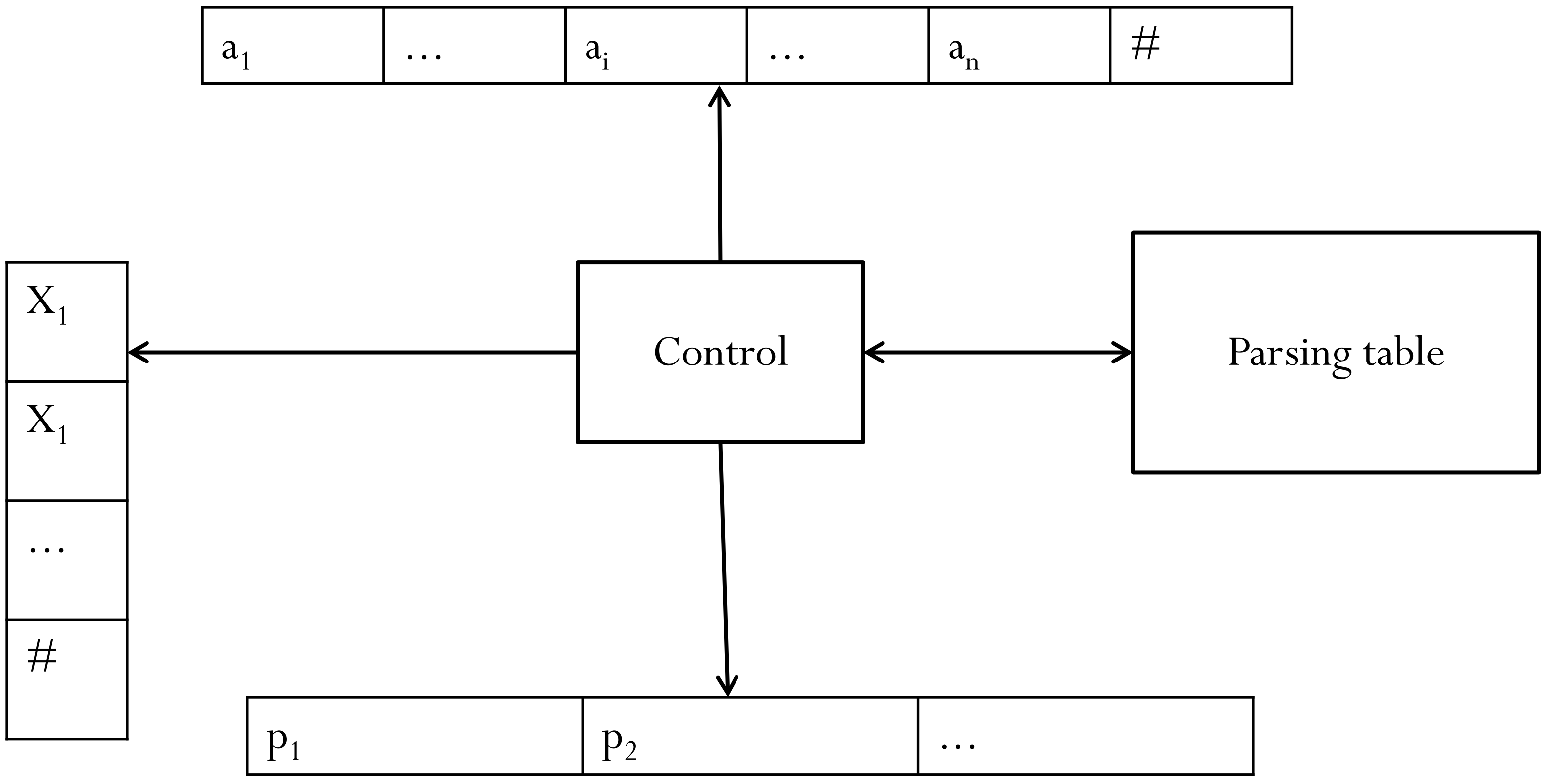
Formal Languages, Automata and Compilers

Course 9

Recap

- Bottom-up syntactic parsing
  - Bottom-up parser
- LR(k) grammars
  - Definition
  - Properties
- LR(0) grammars
  - Characterization theorem for LR(0)
  - LR(0) automaton
- SLR(1) grammars

Bottom-up parser



FIRST and FOLLOW

- $FIRST(\alpha) = \{a | a \in T, \alpha \ {}_{st}\Rightarrow^* au\} \cup$
  if $(\alpha \ {}_{st}\Rightarrow^* \varepsilon)$ then $\{\varepsilon\}$ else $\emptyset$.

- $FOLLOW(A) = \{a | a \in T \cup \{\varepsilon\}, S \ {}_{st}\Rightarrow^* uA\gamma,$
  $a \in FIRST(\gamma)\}$

Calculating FIRST

- 1.for(X ∈ Σ)
  - 2.if(X ∈ T)FIRST(X)={X} else FIRST(X)=∅;
- 3.for(A→aβ ∈P)
  - 4.FIRST(A)=FIRST(A)∪{a};
- 5.FLAG=true;
- 6.while(FLAG){
  - 7.FLAG=false;
  - 8.for(A → $X_1X_2…X_n$ ∈P){
    - 9.i=1;
    - 10.if((FIRST(X1) ⊈ FIRST(A)){
      - 11.FIRST(A)=FIRST(A) ∪(FIRST(X1) – {ε});
      - 12.FLAG=true;
    - 13.}//endif
    - 14.while(i<n&&$X_{i\ st}$⇒* ε)
      - 15.if((FIRST(X_{i+1}) ⊈ FIRST(A)){
        - 16.FIRST(A)=FIRST(A)∪FIRST(X_{i+1});
        - 17.FLAG=true;i++;
      - }//endif
    - }//endwhile
  - }//endfor
- }//endwhile
- for(A∈N)
  - if(A_{st}⇒* ε)FIRST(A)=FIRST(A) ∪{ε};

Calculating FIRST

- **Input:** the grammar G=(N,T,S,P).
- the sets FIRST(X), $X \in \Sigma$.
- $\alpha = X_1X_2 \ldots X_n$, $X_i \in \Sigma, 1 \le i \le n$.
- **Output:** FIRST(α).

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

Example

- Let the grammar G be:
- $S \rightarrow E \mid B$, $E \rightarrow \varepsilon$, $B \rightarrow a \mid$ beginSC end, $C \rightarrow \varepsilon \mid ;\ SC$

- 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)= {;}.

Calculating FOLLOW

- $\varepsilon \in$ FOLLOW(S).
- If $A \rightarrow \alpha B\beta X\gamma \in P$ and $\beta \Rightarrow^{+} \varepsilon$, then FIRST(X) $-\{\varepsilon\} \subseteq$ FOLLOW (B).
  - $S \Rightarrow^{*} \alpha_1 A\, \beta_1 \Rightarrow \alpha_1\alpha B\beta X\gamma\beta_1 \Rightarrow^{*} \alpha_1\alpha BX\gamma\beta_1$ it then follows that FIRST(X)$-\{\varepsilon\} \subseteq$ FOLLOW (B).
- If $A \rightarrow \alpha B\beta \in P$ then FIRST(β)$-\{\varepsilon\} \subseteq$ FOLLOW (B).
- If $A \rightarrow \alpha B\beta \in P$ and $\beta \Rightarrow^{+} \varepsilon$, then FOLLOW(A) $\subseteq$ FOLLOW(B).

Calculating 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

Calculating FOLLOW

- 10.FLAG=true;
- 11.while(FLAG){
  - 12.FLAG=false;
  - 13.for($A \rightarrow X_1X_2 \ldots X_n$){
    - 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

Example

- Let the grammar G be:
- $S \rightarrow E \mid B$, $E \rightarrow \varepsilon$, $B \rightarrow a \mid$ begin SC end, $C \rightarrow \varepsilon \mid ;\ SC$
- FOLLOW(S)=FOLLOW(E)=FOLLOW(B) ={ε, ; , end}
- FOLLOW(C) = {end}.

SLR(1) Grammars

- **Definition**
  - Let G be a grammar for which the LR(0) automaton contains inconsistent states (G is not LR(0)). The grammar G is SLR(1) if for any state t of its LR(0) automaton, the following are true:

  - –If $A\rightarrow\alpha\bullet$, $B \rightarrow \beta\bullet \in t$ then FOLLOW(A) $\cap$FOLLOW(B) = $\emptyset$;
  - –If $A\rightarrow\alpha\bullet$, $B \rightarrow \beta \bullet a\gamma \in t$ then $a \notin$ FOLLOW(A).

- The SLR(1) syntactic analysis is similar to the LR(0) one; the syntactic analysis table has two main components:
  - –The first, ACTION, determines whether the parser will shift or reduce, depending on the state at the top of the stack and the next input symbol
  - –The second, GOTO, determines the state which will be added to the stack following a reduction.

The SLR(1) Parsing Table

- Input:
  - The grammar $G = (N, T, S, P)$ augmented with $S' \to S$;
  - The automaton $M = (Q, \Sigma, g, t_0, Q)$;
  - The sets FOLLOW(A), $A \in V$
- Output:
  - The SLR(1) parsing table, made up of two parts:
  - ACTION(t, a), $t \in Q$, $a \in T \cup \{ \# \}$,
  - GOTO(t, A), $t \in Q$, $A \in N$.

The SLR(1) Parsing Table

- for(t ∈ Q)
  - for (a ∈ T) ACTION(t, a) = "error";
  - for (A ∈ V) GOTO(t, A) = "error";
- for(t ∈ Q}{
  - for(A→ α·aβ ∈ t)
    - ACTION(t,a)="S g(t, a)";//shift to g(t, a)
  - for(B→ γ· ∈ t){// accept or reduce
    - if(B == 'S') ACTION(t, #) = "accept";
    - else
      - for(a ∈ FOLLOW(B)) ACTION(t,a)="R B→γ";
  - }// endfor
  - for (A ∈ N) GOTO(t, A) = g(t, A);
- }//endfor

SLR(1) Parsing

- **shift**: $(\sigma t,\ au\#,\ \pi) \vdash (\sigma t t',\ u\#,\ \pi)$ if ACTION(t, a)=St';
- **reduce**: $(\sigma t \sigma' t',\ u\#,\ \pi) \vdash (\ \sigma t t'',\ u\#,\ \pi r)$ ACTION(t, a) = Rp where p= $A \to \beta$, $|\sigma' t'| = |\beta|$ and t''= GOTO(t, A);
- **accept**: $(t_0 t,\ \#,\ \pi)$ if ACTION(t,a) = "accept"; The analyzer stops and accepts the analyzed word and π is the parse for the word (the sequence of productions, in reverse, for the rightmost derivation of w).
- **error**: $(\sigma\ t,\ au\#,\ \pi) \vdash$ error if ACTION(t,a) = "error"; The analyzer stops and rejects the analyzed word.

SLR(1) Parsing

- Input:
  - The grammar G = (N, T, S, P) which is SLR(1) ;
  - The SLR(1) parsing table (ACTION, GOTO);
  - The input word $w \in T^*$.
- Output:
  - The bottom-up syntactic parse of w, if $w \in L(G)$;
  - error, otherwise.
- The analyzer uses the stack St for performing the shift/reduce transitions

SLR(1) Parsing

- `char ps[] = "w#"; //ps is the input word w`
- `int i = 0; // the current position of the input letter`
- `St.push(t0); // initialize the stack with t0`
- `while(true) { // repeat until success or error`
  - `t = St.top();`
  - `a = ps[i] // a is the current input symbol`
  - `if(ACTION(t,a) == "accept") exit("accept");`
  - `if(ACTION(t,a) == "Dt'"){`
    - `St.push(t');`
    - `i++; // move forward in w`
  - `}//endif`
  - `else {`
    - `if(ACTION(t,a) ==` "R $A \rightarrow X_1X_2\ldots X_m$"`){`
      - `for( i = 1; i ≤m; i++) St.pop();`
      - `St.push(GOTO(St.top, A));`
    - `} //endif`
    - `else exit("error");`
  - `}//endelse`
- `}//endwhile`

Example

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

0

S → •E
E → •E+T
E → •T
T → •T*F
T → •F
F → •(E)
F → •a

1

S → E•
E → E•+T

2

E → T•
T → T•*F

3

T → F•

4

F → (•E)
E → •E+T
E → •T
T → •T*F
T → •F
F → •(E)
F → •a

5

F → a•

6

E → E+•T
T → •T*F
T → •F
F → •(E)
F → •a

7

T → T*•F
F → •(E)
F → •a

8

F → (E•)
E → E•+T

9

E → E+T•
T → T•*F

10

T → T*F•

11

F → (E)•

Transition Table for the LR(0) Automaton

	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		6			11			
9			7					
10								
11								

SLR(1) Test

- G is not LR(0), as the states 1, 2, 9 contain shift/reduce conflicts

- FOLLOW(S)={#}, FOLLOW(E)={#,+,)}

- The grammar is SLR(1) because:
  - In state 1: + $\notin$ FOLLOW(S);
  - In state 2: * $\notin$ FOLLOW(E);
  - In state 9: * $\notin$ FOLLOW(E).

SLR(1) Parsing Table

State	ACTION						GOTO		
	a	+	*	(	)	#	E	T	F
0	S5			S4			1	2	3
1		S6				accept			
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	Output
0	a*(a+a)#	shift	
05	*(a+a)#	reduce	6.F $\longrightarrow$ a
03	*(a+a)#	reduce	4.T $\longrightarrow$ F
02	*(a+a)#	shift	
027	(a+a)#	shift	
0274	a+a)#	shift	
02745	+a)#	reduce	6.F $\longrightarrow$ a
02743	+a)#	reduce	4.T $\longrightarrow$ F
02742	+a)#	reduce	2.E $\longrightarrow$ T
02748	+a)#	shift	

Stack	Input	Action	Output
027486	a)#	shift	
0274865	)#	reduce	6.F $\rightarrow$ a
0274863	)#	reduce	4.T $\rightarrow$ F
0274869	)#	reduce	1.E $\rightarrow$ E+T
02748	)#	shift	
02748(11)	#	reduce	5.F $\rightarrow$(E)
027(10)	#	reduce	3.T $\rightarrow$ T*F
02	#	reduce	2.E $\rightarrow$ T
01	#	accept	

LR(1) Gramammars

- **Definition**
  - Let G = (V, T, S, P) be a reduced grammar. An LR(1) article for the grammar G is a pair $(A \rightarrow \alpha\cdot\beta, a)$, where $A\rightarrow\alpha\cdot\beta$ is an LR(0) article, and $a \in FOLLOW(A)$ (# instead of ε).
- **Definition**
  - The article $(A \rightarrow \beta1 \cdot \beta2, a)$ is valid for the viable prefix $\alpha\beta1$ if the following are true
    - $S\ r\Rightarrow^{*}\alpha Au \Rightarrow \alpha\beta1\beta2u$
    - and a = 1:u (a = # if u = ε).
- **Theorem**
  - A grammar G = (N, T, S, P) is LR(1) iff for any viable prefix φ, there are no two distinct articles, valid for φ, with $(A \rightarrow \alpha\cdot, a)$, $(B \rightarrow\beta\cdot\gamma, b)$ where $a \in FIRST(\gamma b)$.

LR(1) Gramammars

- There are no shift/reduce conflicts. Such a conflict would mean that two articles $(A \to \alpha\cdot, a)$ and $(B \to \beta\cdot a\beta', b)$ are both valid for the same prefix.
- There are no reduce/reduce conflicts. Such a conflict would mean that two complete articles $(A \to \alpha\cdot, a)$ and $(B \to \beta\cdot, a)$ are both valid for the same prefix.
- To check if a grammar is LR(1) we build the LR(1) automaton in a similar manner to the LR(0) automaton:
  - States contain sets of LR(1) articles
  - Transitions are made by reading symbols to the right of the point
  - Closure is based on the fact that, if the article $(B \to \beta\cdot A\beta', b)$ is valid for the viable prefix φ then all articles of the form $(A \to\cdot\alpha, a)$ where $a \in FIRTST(\alpha a)$ are valid for the same prefix.

Closure procedure for 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;

LR(1) Automaton

- t0 = closure((S′ → ·S,#));T={t_0};marked(t_0)=false;
- while(∃ t∈T&& !marked(t)){ // marked(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′ = closure(t′);
        - if(t′∈ T) {
          - T= T ∪{ t′ };
          - marked(t′) = false;
        - }//endif
        - g(t, X) = t′;
      - } //endif
  - } //endfor
  - marked(t) = true;
- } // endwhile

LR(1) Automaton

- **Theorem**
  - The automaton M built in algorithm 2 is deterministic and L(M) is the set of viable prefixes for G. Moreover, for any viable prefix γ, $g(t_0,\gamma)$ is the set of LR(1) articles valid for γ.
- The LR(1) automaton can be used to check if G is LR(1)
  - Reduce/reduce conflict: If a state t contains articles of the form $(A \rightarrow \alpha\cdot, a)$, $(B \rightarrow \beta\cdot, a)$ then G is not LR(1);
  - Shift/reduce conflict: If a state t contains articles of the form $(A \rightarrow \alpha\cdot, a)$ and $(B \rightarrow \beta_1\cdot a\beta_2, b)$, then G is not LR(1).
  - G is LR(1) if all the states $t \in T$ are free of conflicts

Example

- S ⟶ L=R | R, L ⟶ *R | a, R ⟶ L

0

(S' → •S, #)
(S → •L=R, #)
(S → •R, #)
(L → •*R, {=,#})
(L → •a, {=,#})
(R → •L, #)

1

(S' → S•, #)

2

(S → L•=R, #)
(R → L•, #)

3

(S → R•, #)

4

(L → *•R, {=, #})
(R → •L, {=, #})
(L → •*R, {=, #})
(L → •a, {=, #})

5

(L → a•, {=,#})

6

(S → L=•R, #)
(R → •L, #)
(L → •*R, #)
(L → •a, #)

7

(L → *R•, {=,#})

8

(R → L•, {=,#})

9

(S → L=R• , #)

10

(R → L•, #)

11

(L → *•R, #)
(R → •L, #)
(L → •*R, #)
(L → •a, #)

12

(L → a•, #)

13

(L → *R•, #)

Transition Table

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						

LR(1) Parsing Table

- for(t ∈ T)
  - for (a ∈ T) ACTION(t, a) = “error”;
  - for (A ∈ V) GOTO(t, A) = “error”;
- for(t ∈ T}{
  - for((A→ α·aβ, L) ∈ t)
    - ACTION(t,a)=“S g(t, a)”;//Shift to g(t, a)
  - for((B→ γ·, L) ∈ t){// accept or reduce
    - for(c ∈ L) {
      - if(B == ‘S’) ACTION(t, #) = “accept”;
      - else ACTION(t,c)=“R B→γ”;//Reduce with B→γ
      - }//endfor
  - }// endfor
  - for (A ∈ N) GOTO(t, A) = g(t, A);
- }//endfor

- 0:S'→S, 1:S →L=R, 2:S →R, 3:L →*R, 4:L →a, 5:R →L

State	ACTION				GOTO		
	a	=	*	#	S	L	R
0	S5		S4		1	2	3
1				Acc			
2		S6		R5			
3				R2			
4	S5		S4			8	7
5		R4		R4			
6	S12		S11			10	9
7		R3		R3			
8		R5		R5			
9				R1			
10				R5			
11	S12		S11			10	13
12				R4			
13				R3			

Example

- For the words
  - ***a
  - a=**a
  - *a=**a
- What is the LR(1) analysis?

LALR(1) Gramammars

- **Definition**
  - Let t be a state in the LR(1) automaton for G. The nucleus of this state is the set of LR(0) articles which make up the first component of the LR(1) articles in t.

- **Definition**
  - Two states t_1 and t_2 of the LR(1) automaton for G are equivalent if they have the same nucleus.

LALR(1) Grammars

- Each state of the LR(1) automaton is a set of LR(1) articles. Starting with two states t_1 şi t_2 we can define the state t1 ∪ t2.
  - Let $t_1 = \{(L \rightarrow *R\cdot, \{=, \#\})\}$, $t_2 = \{(L \rightarrow *R\cdot, \#)\}$, then $t_1 \cup t_2 = t_1$ as $t_2 \subset t_1$.

- **Definition**
  - Let G be an LR(1) grammar and $M = (Q, \Sigma, g, t_0, Q)$ its corresponding LR(1) automaton. We say that G is LALR(1) (Look Ahead LR(1)) if for any pair of equivalent states t_1, t_2 from the LR(1) automaton, the state $t_1 \cup t_2$ is free of conflicts.

LALR(1) Parsing Table

- **Input**: G = (N, T, S, P) augmented with S' → S;
- **Output**: LALR(1) parsing table (ACTION and GOTO).
- **Algorithm**:
  - **1.** Build the LR(1) automaton, M = (Q, Σ, g, t_0, Q). Let Q = {t_0, t_1,…, t_n}. If all the states of Q are free of conflict step 2, else stop (G is not LR(1)).
  - **2.** Determine the equivalent states of Q and perform unions over them. This results into a new set of states, Q' = {t'_0, t'_1,…, t'_m}
  - **3.** If Q' contains states which are not free of conflict, stop (G is not LALR(1)).

LALR(1) Parsing Table

- **4.** Build the automaton M' = (Q', Σ, g', t'0, Q'), where ∀ t'∈Q':
  - **5.** If t' ∈ Q then g'(t', X) = g(t, X), ∀X∈Σ;
  - **6.** If $t' = t_1 \cup t_2 \cup \ldots$, $t_1, t_2, \ldots \in Q$, then
    - **7.** g'(t', X) = g(t1, X)∪g(t2, X)∪….
- **8.** Build the parsing table from the automaton M' using the algorithm for building the LR(1) parsing table. The table thus obtained is called the LALR(1) table for the grammar G.

Example

- For the previously discussed grammar, we have 4U11 = 4, 5U12 = 5, 7U13 = 7, 8U10 = 8

State	ACTION				GOTO		
	a	*	=	#	S	L	R
0	S5		S4		1	2	3
1				Acc			
2		S6		R5			
3				R2			
4	S5		S4			8	7
5		R4					
6	S5		S4			8	9
7		R3		R3			
8		R5		R5			
9				R1			

Example

- Not all LR(1) grammars are also LALR(1).

  - $S \rightarrow aAb \mid bAd \mid aBd \mid bBb$
  - $A \rightarrow e$
  - $B \rightarrow e$

Bibliography

- A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Second Edition. Addison-Wesley, 2007
- G. Grigoraş, *Construcția compilatoarelor. Algoritmi fundamentali*, Editura Universității "Alexandru Ioan Cuza", Iaşi, 2005