Compiler Construction
Bottom-Up Parsing
LR Parsing

The Canonical LR(0) Collection -- Example

I_0: E' → .E
E → .E+T
E → .T
T → .T*F
T → .F
F → .(E)
F → .id

I_1: E' → E.
E → E.+T

I_2: E → T.
T → T.*F

I_3: T → F.

I_4: F → (.E)
E → .E+T
E → .T
T → .T*F
T → .F
F → .(E)
F → .id

I_5: F → id.

I_6: E → E+.T
T → .T*F
T → .F
F → .(E)
F → .id

I_7: T → T*.F
F → .(E)
F → .id

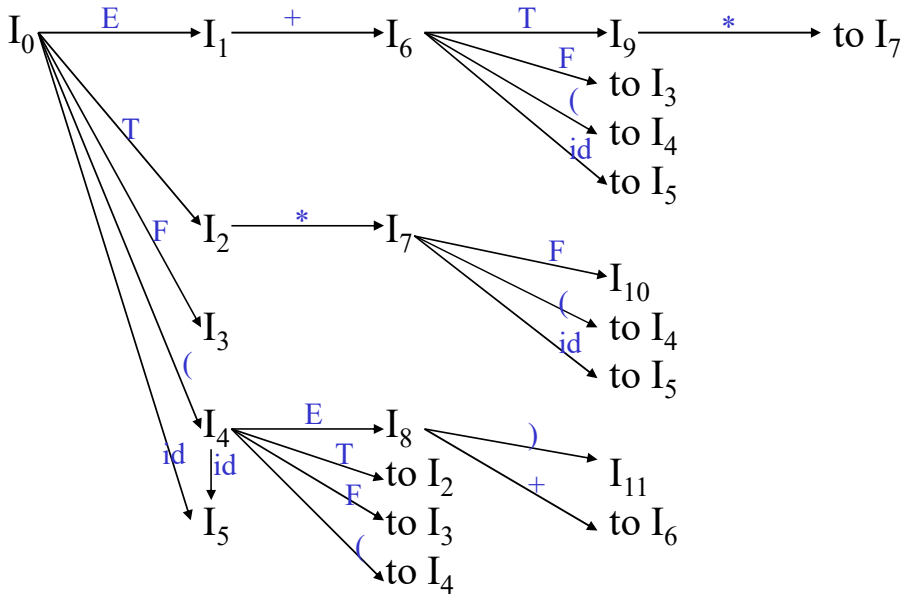
I_8: F → (E.)
E → E.+T

I_9: E → E+T.
T → T.*F

I_{10}: T → T*F.

I_{11}: F → (E).

Transition Diagram (DFA) of Goto Function



Constructing SLR Parsing Table

(of an augumented grammar G')

1. Construct the canonical collection of sets of LR(0) items for G'. $\mathbf{C \leftarrow \{I_0,...,I_n\}}$
2. Create the parsing action table as follows
   - If a is a terminal, $\mathbf{A \rightarrow \alpha.a\beta}$ in I_i and goto(I_i,a)=I_j then action[i,a] is ***shift j.***
   - If $\mathbf{A \rightarrow \alpha.}$ is in I_i, then **action[i,a]** is ***reduce*** $A \rightarrow \alpha$ for all a in **FOLLOW(A)** where **A≠S'**.
   - If $\mathbf{S' \rightarrow S.}$ is in I_i, then action[i,$] is ***accept***.
   - If any conflicting actions generated by these rules, the grammar is not SLR(1).
3. Create the parsing goto table
   - for all non-terminals A, if goto(I_i,A)=I_j then goto[i,A]=j
4. All entries not defined by (2) and (3) are errors.
5. Initial state of the parser contains $S' \rightarrow .S$

Parsing Tables of Expression Grammar

	Action Table						Goto Table		
state	**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			

SLR(1) Grammar

- An LR parser using SLR(1) parsing tables for a grammar G is called as the SLR(1) parser for G.
- If a grammar G has an SLR(1) parsing table, it is called SLR(1) grammar (or SLR grammar in short).
- Every SLR grammar is unambiguous, but every unambiguous grammar is not a SLR grammar.

Example 2

$S \rightarrow L=R$

$S \rightarrow R$

$L \rightarrow *R$

$L \rightarrow id$

$R \rightarrow L$

I_0: $S' \rightarrow .S$

$S \rightarrow .L=R$

$S \rightarrow .R$

$L \rightarrow .*R$

$L \rightarrow .id$

$R \rightarrow .L$

Conflict Example

S → L=R
S → R
L→ *R
L → id
R → L

I_0: S' → .S
S → .L=R
S → .R
L → .*R
L → .id
R → .L

I_1: S' → S.

I_2: S → L.=R
R → L.

I_3: S → R.

I_4: L → *.R
R → .L
L→ .*R
L → .id

I_5: L → id.

I_6: S → L=.R
R → .L
L→ .*R
L → .id

I_7: L → *R.

I_8: R → L.

I_9: S → L=R.

Problem

FOLLOW(R)={=,$}

= → shift 6

= → reduce by R → L

shift/reduce conflict

Action[2,=] = shift 6

Action[2,=] = reduce by R → L

[S ⇒L=R ⇒*R=R] so follow(R) contains, =

shift/reduce and reduce/reduce conflicts

- If a state does not know whether it will make a shift operation or reduction for a terminal, we say that there is a **shift/reduce conflict**.
- If a state does not know whether it will make a reduction operation using the production rule `i` or `j` for a terminal, we say that there is a **reduce/reduce conflict**.
- If the SLR parsing table of a grammar G has a conflict, we say that that grammar is not SLR grammar.

Conflict Example2

S → AaAb
S → BbBa
A → ε
B → ε

I_0: S' → .S
S → .AaAb
S → .BbBa
A → .
B → .

Problem

FOLLOW(A)={a,b}
FOLLOW(B)={a,b}

a → reduce by A → ε
a → reduce by B → ε
reduce/reduce conflict

b → reduce by A → ε
b → reduce by B → ε
reduce/reduce conflict