Constructing SLR Parsing Tables – LR(0) Item

- An **LR(0) item** of a grammar G is a production of G a dot at the some position of the right side.
- Ex: $A \rightarrow aBb$ Possible LR(0) Items: $A \rightarrow \bullet aBb$
  (four different possibility) $A \rightarrow a \bullet Bb$
  $A \rightarrow aB \bullet b$
  $A \rightarrow aBb \bullet$
- Sets of LR(0) items will be the states of action and goto table of the SLR parser.
- A collection of sets of LR(0) items (**the canonical LR(0) collection**) is the basis for constructing SLR parsers.
- *Augmented Grammar*:

  G' is G with a new production rule $S' \rightarrow S$ where S' is the new starting symbol.

The Closure Operation

- If ***I*** is a set of LR(0) items for a grammar G, then ***closure(I)*** is the set of LR(0) items constructed from I by the two rules:

1. Initially, every LR(0) item in I is added to closure(I).
2. If $A \rightarrow \alpha \bullet B\beta$ is in closure(I) and $B \rightarrow \gamma$ is a production rule of G; then $B \rightarrow \bullet \gamma$ will be in the closure(I).
   We will apply this rule until no more new LR(0) items can be added to closure(I).

The Closure Operation -- Example

$E' \to E$

$E \to E+T$

$E \to T$

$T \to T*F$

$T \to F$

$F \to (E)$

$F \to id$

closure($\{E' \to \bullet E\}$) =

$\{ E' \to \bullet E$ ← kernel items

$E \to \bullet E+T$

$E \to \bullet T$

$T \to \bullet T*F$

$T \to \bullet F$

$F \to \bullet (E)$

$F \to \bullet id \}$

Goto Operation

- If I is a set of LR(0) items and X is a grammar symbol (terminal or non-terminal), then goto(I,X) is defined as follows:
  - If $A \rightarrow \alpha \bullet X\beta$ in I
    then every item in **closure({$A \rightarrow \alpha X \bullet \beta$})** will be in goto(I,X).

Example:

I ={ $E' \rightarrow \bullet E$, $E \rightarrow \bullet E+T$, $E \rightarrow \bullet T$,
$T \rightarrow \bullet T*F$, $T \rightarrow \bullet F$,
$F \rightarrow \bullet (E)$, $F \rightarrow \bullet id$ }

goto(I,E) = { $E' \rightarrow E \bullet$, $E \rightarrow E \bullet +T$ }

goto(I,T) = { $E \rightarrow T \bullet$, $T \rightarrow T \bullet *F$ }

goto(I,F) = { $T \rightarrow F \bullet$ }

goto(I,() = { $F \rightarrow (\bullet E)$, $E \rightarrow \bullet E+T$, $E \rightarrow \bullet T$, $T \rightarrow \bullet T*F$, $T \rightarrow \bullet F$,
$F \rightarrow \bullet (E)$, $F \rightarrow \bullet id$ }

goto(I,id) = { $F \rightarrow id \bullet$ }

Construction of The Canonical LR(0) Collection

- To create the SLR parsing tables for a grammar G, we will create the canonical LR(0) collection of the grammar G'.

- ***Algorithm***:

  C is { closure({S'→•S}) }

  repeat the followings until no more set of LR(0) items can be added to ***C***.

      for each I in ***C*** and each grammar symbol X

          if goto(I,X) is not empty and not in ***C***

              add goto(I,X) to ***C***

- goto function is a DFA on the sets in C.

The Canonical LR(0) Collection -- Example

I_0: $E' \to .E$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

I_1: $E' \to E.$
$E \to E.+T$

I_2: $E \to T.$
$T \to T.*F$

I_3: $T \to F.$

I_4: $F \to (.E)$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

I_5: $F \to id.$

I_6: $E \to E+.T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

I_7: $T \to T*.F$
$F \to .(E)$
$F \to .id$

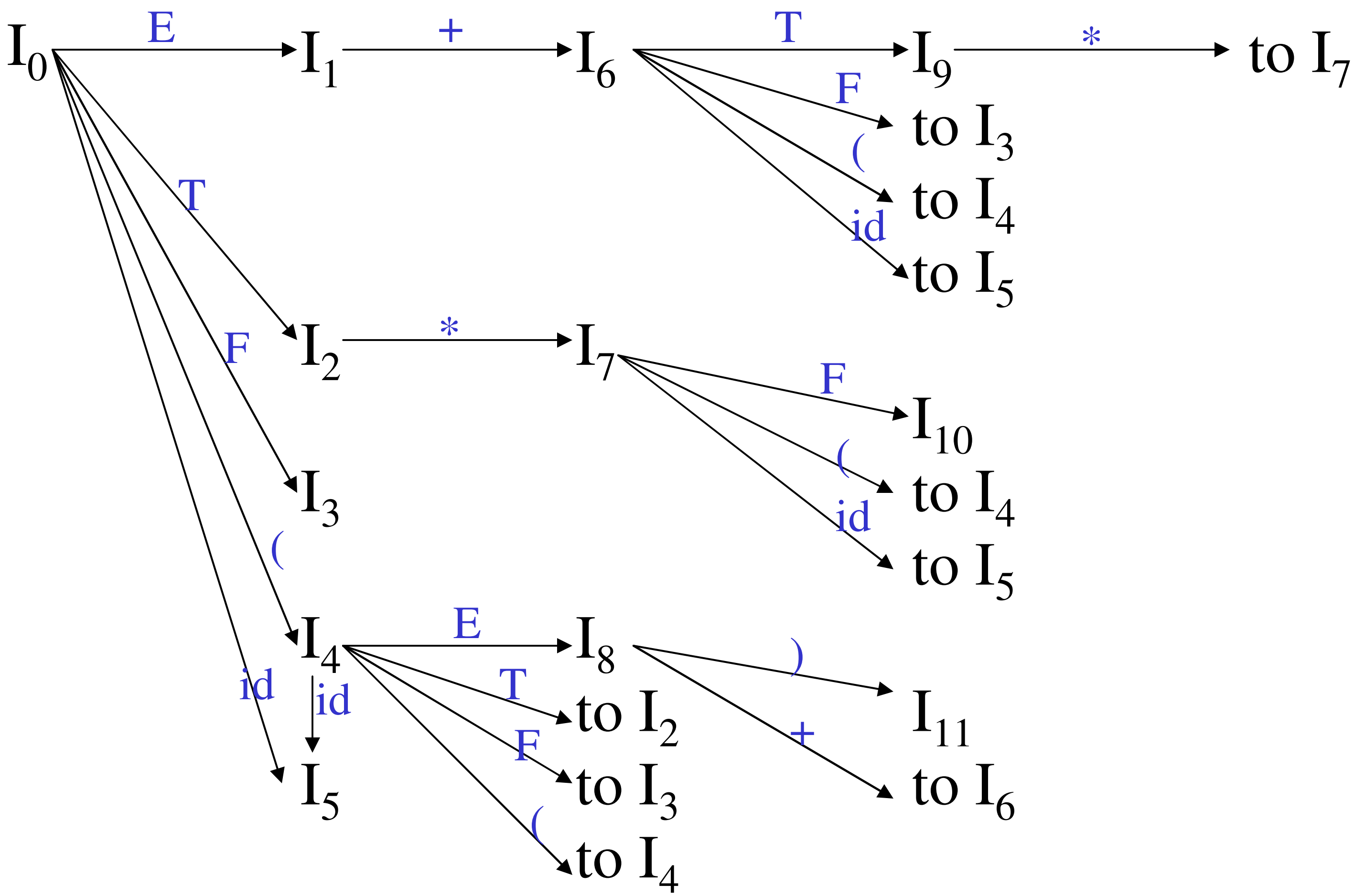
I_8: $F \to (E.)$
$E \to E.+T$

I_9: $E \to E+T.$
$T \to T.*F$

I_{10}: $T \to T*F.$

I_{11}: $F \to (E).$

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'.
   $C \leftarrow \{I_0,...,I_n\}$

2. Create the parsing action table as follows
   - If a is a terminal, $A \rightarrow \alpha.a\beta$ in I_i and goto(I_i,a)=I_j then action[i,a] is ***shift j.***
   - If $A \rightarrow \alpha.$ is in I_i , then action[i,a] is ***reduce*** $\boldsymbol{A \rightarrow \alpha}$ for all a in FOLLOW(A) where $A \neq S'$.
   - If $S' \rightarrow S.$ 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.

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 Example

$S \rightarrow L=R$
$S \rightarrow R$
$L \rightarrow *R$
$L \rightarrow id$
$R \rightarrow L$

I_0: $S' \rightarrow .S$
$S \rightarrow .L=R$
$S \rightarrow .R$
$L \rightarrow .*R$
$L \rightarrow .id$
$R \rightarrow .L$

I_1: $S' \rightarrow S.$

I_2: $S \rightarrow L.=R$
$R \rightarrow L.$

I_3: $S \rightarrow R.$

I_4: $L \rightarrow *.R$
$R \rightarrow .L$
$L \rightarrow .*R$
$L \rightarrow .id$

I_5: $L \rightarrow id.$

I_6: $S \rightarrow L=.R$
$R \rightarrow .L$
$L \rightarrow .*R$
$L \rightarrow .id$

I_7: $L \rightarrow *R.$

I_8: $R \rightarrow L.$

I_9: $S \rightarrow L=R.$

Problem
FOLLOW(R)={=,$}
= → shift 6
→ reduce by $R \rightarrow L$
shift/reduce conflict

Conflict Example2

$S \rightarrow AaAb$

$S \rightarrow BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

I_0: $S' \rightarrow .S$

$S \rightarrow .AaAb$

$S \rightarrow .BbBa$

$A \rightarrow .$

$B \rightarrow .$

Problem

FOLLOW(A)={a,b}

FOLLOW(B)={a,b}

a → reduce by $A \rightarrow \varepsilon$

a → reduce by $B \rightarrow \varepsilon$

reduce/reduce conflict

b → reduce by $A \rightarrow \varepsilon$

b → reduce by $B \rightarrow \varepsilon$

reduce/reduce conflict