CSE304 Compiler Design

Syntax Analysis (LR, LALR Parsers)

YOUNGMIN KWON / TONY MIONE

A Non-SLR(1) Grammar

$$\begin{array}{lcl} S & \rightarrow & L = R \mid R \\ L & \rightarrow & *R \mid \textbf{id} \\ R & \rightarrow & L \end{array}$$

I_2 has a shift/reduce conflict:

- S->L.=R : action[2,=] shift 6
- R->L. : action[2,=] reduce "R->L"
  - (= is in FOLLOW(R): S => L=R => *R=R)
- I_2 is for a viable prefix L only and should not reduce R->L.

$$\begin{array}{ll} I_0: & S' \rightarrow \cdot S \\ & S \rightarrow \cdot L = R \\ & S \rightarrow \cdot R \\ & L \rightarrow \cdot * R \\ & L \rightarrow \cdot \textbf{id} \\ & R \rightarrow \cdot L \end{array}$$

$$I_1: \quad S' \rightarrow S \cdot$$

$$\begin{array}{ll} I_2: & S \rightarrow L \cdot = R \\ & R \rightarrow L \cdot \end{array}$$

$$I_3: \quad S \rightarrow R \cdot$$

$$\begin{array}{ll} I_4: & L \rightarrow * \cdot R \\ & R \rightarrow \cdot L \\ & L \rightarrow \cdot * R \\ & L \rightarrow \cdot \textbf{id} \end{array}$$

$$I_5: \quad L \rightarrow \textbf{id} \cdot$$

$$\begin{array}{ll} I_6: & S \rightarrow L = \cdot R \\ & R \rightarrow \cdot L \\ & L \rightarrow \cdot * R \\ & L \rightarrow \cdot \textbf{id} \end{array}$$

$$I_7: \quad L \rightarrow * R \cdot$$

$$I_8: \quad R \rightarrow L \cdot$$

$$I_9: \quad S \rightarrow L = R \cdot$$

LR Parsing Table

Add more information to the states

Split states to indicate which input symbol can follow the handle.

LR(1) item

- [A->α.β, a], where A->αβ is a production and a (lookahead of the item) is a terminal or $.
- Lookahead has no effect on the item [A->α.β, a] unless β is ϵ.
- For [A->α., a], call for the reduction only if the next input symbol is a.

LR(1) item [A->α.β, a] is valid for a viable prefix γ if there is a derivation $S \Rightarrow^* \delta A w \Rightarrow \delta\alpha\beta w$, where

- $\gamma = \delta\alpha$
- Either a is the first symbol of w or w is ϵ and a is $.

LR Parsing Table

Changes to CLOSURE

- LR(0) items: add [B->.η] to I if [A->α.Bβ] is in I.
- LR(1) items: add [B->.η, b] to I if [A->α.Bβ, a] is in I and b is a terminal in FIRST(βa).
- Why b is a terminal in FIRST(βa)
  - Suppose that S =>* δAax => δαBβax
  - For the same viable prefix (δα), S =>* δαBby => δαηby
  - b can be FIRST(β) or a if β =>* ϵ. Hence, b can be FIRST(βa)

LR(1) items

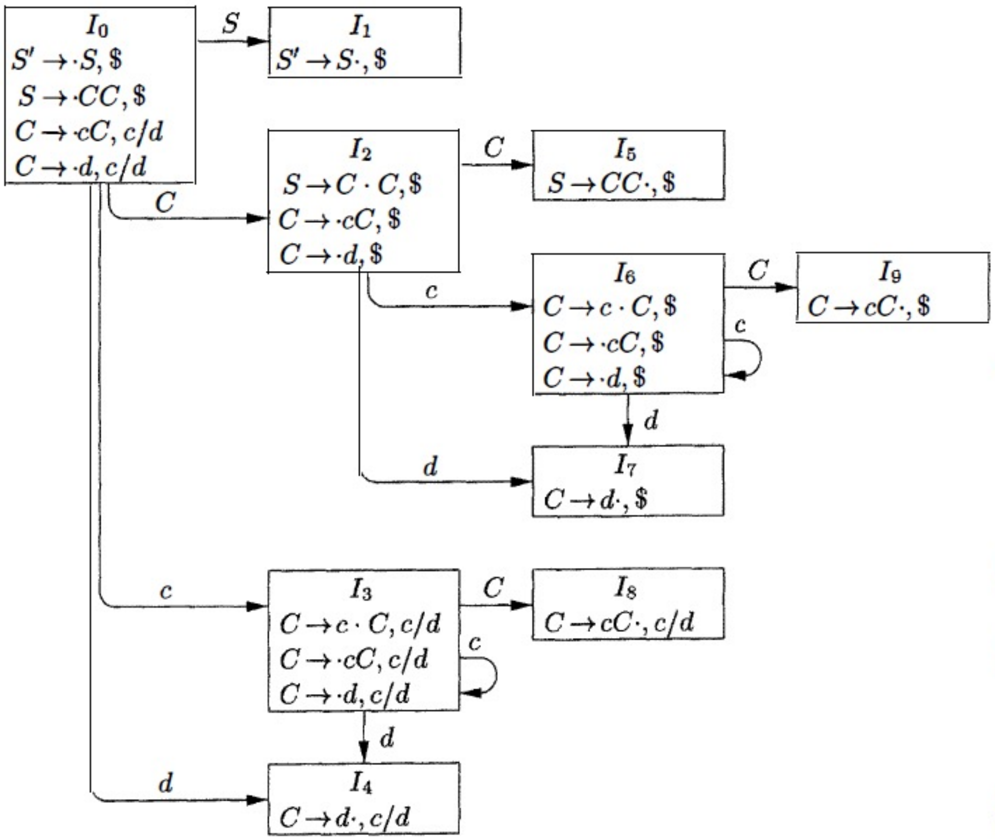
```
SetOfItems CLOSURE(I) {
    repeat
        for ( each item [A → α·Bβ, a] in I )
            for ( each production B → γ in G' )
                for ( each terminal b in FIRST(βa) )
                    add [B → ·γ, b] to set I;
    until no more items are added to I;
    return I;
}

SetOfItems GOTO(I, X) {
    initialize J to be the empty set;
    for ( each item [A → α·Xβ, a] in I )
        add item [A → αX·β, a] to set J;
    return CLOSURE(J);
}
```

LR(1) Items

```
void items(G') {
    initialize C to CLOSURE({[S' → ·S, $]});
    repeat
        for ( each set of items I in C )
            for ( each grammar symbol X )
                if ( GOTO(I, X) is not empty and not in C )
                    add GOTO(I, X) to C;
    until no new sets of items are added to C;
}
```

LR(1) Items Example



$$
\begin{aligned}
S' &\to S \\
S &\to C\ C \\
C &\to c\ C \mid d
\end{aligned}
$$

I_0 : $S \to \cdot S, \$$
$S \to \cdot CC,\ \$$
$C \to \cdot cC,\ c/d$
$C \to \cdot d,\ c/d$

I_1 : $S' \to S\cdot,\ \$$

I_2 : $S \to C\cdot C,\ \$$
$C \to \cdot cC,\ \$$
$C \to \cdot d,\ \$$

I_3 : $C \to c\cdot C,\ c/d$
$C \to \cdot cC,\ c/d$
$C \to \cdot d,\ c/d$

I_4 : $C \to d\cdot,\ c/d$

I_5 : $S \to CC\cdot, \$$

I_6 : $C \to c\cdot C,\ \$$
$C \to \cdot cC,\ \$$
$C \to \cdot d,\ \$$

I_7 : $C \to d\cdot,\ \$$

I_8 : $C \to cC\cdot,\ c/d$

I_9 : $C \to cC\cdot,\ \$$

Constructing LR Parsing Table

1. Construct $C' = \{I_0, I_1, \cdots, I_n\}$, the collection of sets of LR(1) items for G'.

2. State i of the parser is constructed from I_i. The parsing action for state i is determined as follows.

   (a) If $[A \to \alpha \cdot a\beta, b]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set $\text{ACTION}[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha \cdot, a]$ is in I_i, $A \neq S'$, then set $\text{ACTION}[i, a]$ to "reduce $A \to \alpha$."

   (c) If $[S' \to S \cdot, \$]$ is in I_i, then set $\text{ACTION}[i, \$]$ to "accept."

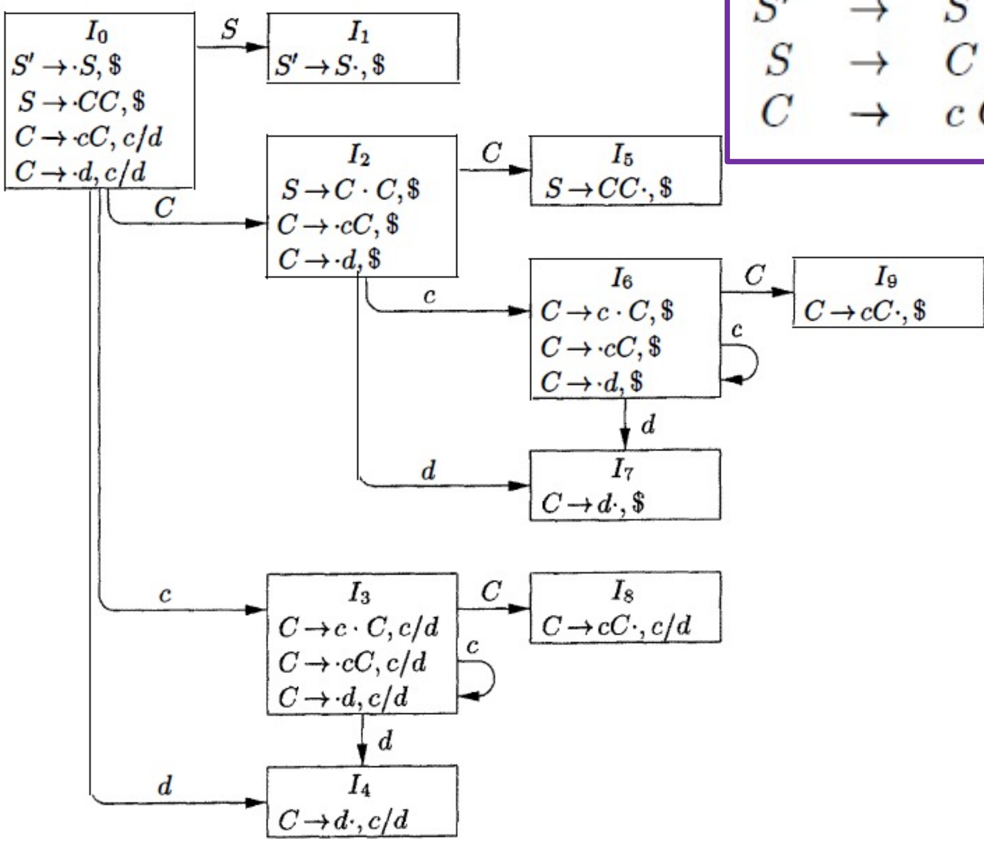
   If any conflicting actions result from the above rules, we say the grammar is not LR(1). The algorithm fails to produce a parser in this case.

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then $\text{GOTO}[i, A] = j$.

4. All entries not defined by rules (2) and (3) are made "error."

5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S, \$]$.

Constructing LR Parsing Table



$$
\begin{aligned}
S' &\to S \\
S &\to C\ C \\
C &\to c\ C \mid d
\end{aligned}
$$

STATE	ACTION			GOTO	
	c	d	\$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

LALR Parsing Table

Merge LR(1) items with the same core (first component).

No shift/reduce conflicts are introduced by the merge:

- Suppose there is a conflict in a merged state.
- There are [A->α.,a] and [B->β.aγ, b] in the item.
- Because the cores are the same, before the merge there is an item with [A->α.,a] and [B->β.aγ, c].
- Hence, the original item before the merge has a shift/reduce conflict.

LALR Parsing Table

A reduce/reduce conflict can be introduced by the merge.

Exercise:

- Find LR(1) items for the grammar below
- Check how the reduce/reduce conflict is introduced by the merge.

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow a\ A\ d \mid b\ B\ d \mid a\ B\ e \mid b\ A\ e \\
A &\rightarrow c \\
B &\rightarrow c
\end{aligned}
$$

LALR Parsing Table Construction

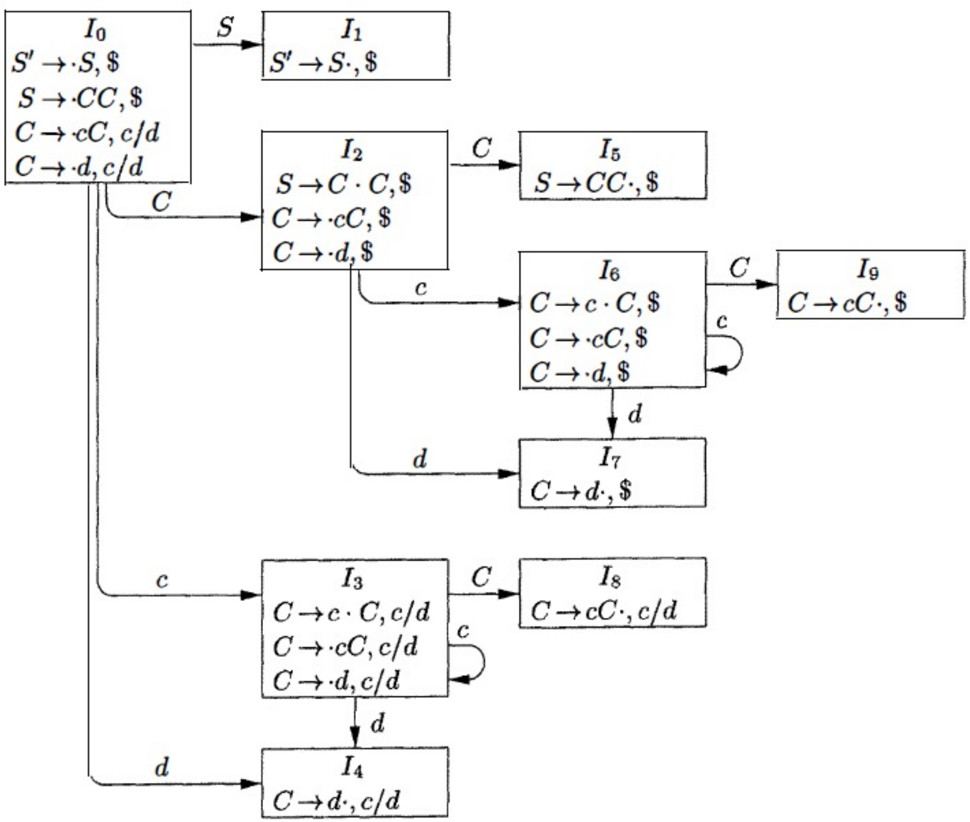
1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(1) items.

2. For each core present among the set of LR(1) items, find all sets having that core, and replace these sets by their union.

3. Let $C' = \{J_0, J_1, \ldots, J_m\}$ be the resulting sets of LR(1) items. The parsing actions for state i are constructed from J_i in the same manner as in Algorithm 4.56. If there is a parsing action conflict, the algorithm fails to produce a parser, and the grammar is said not to be LALR(1).

4. The GOTO table is constructed as follows. If J is the union of one or more sets of LR(1) items, that is, $J = I_1 \cap I_2 \cap \cdots \cap I_k$, then the cores of $\text{GOTO}(I_1, X)$, $\text{GOTO}(I_2, X), \ldots, \text{GOTO}(I_k, X)$ are the same, since $I_1, I_2, \ldots, I_k$ all have the same core. Let K be the union of all sets of items having the same core as $\text{GOTO}(I_1, X)$. Then $\text{GOTO}(J, X) = K$.

LALR Parsing Table Example



$$
\begin{aligned}
S' &\to S \\
S &\to C\ C \\
C &\to c\ C \mid d
\end{aligned}
$$

STATE	ACTION			GOTO	
	c	d	\$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

LR Parser and LALR Parser

LR parser and LALR parser mimic each other for the correct input.

For erroneous input,

- LR parser detects error immediately.
- LALR parser reduces several more steps and detects an error before shifting any symbols.

Exercise:

1. Compare the steps for cdcd.
2. Compare the steps for ccd.

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		