CS 4300: Compiler Theory

Chapter 4
Syntax Analysis

Dr. Xuejun Liang

Outlines (Sections)

1. Introduction
2. Context-Free Grammars
3. Writing a Grammar
4. Top-Down Parsing
5. Bottom-Up Parsing
6. Introduction to LR Parsing: Simple LR
7. More Powerful LR Parsers
8. Using Ambiguous Grammars
9. Parser Generators

Quick Review of Last Lecture

- LR Parsing
  - Model of an LR Parser
  - LR Parsing Driver
  - Example LR(0) Parsing Table
- SLR: Simple extension of LR(0) shift-reduce parsing
  - Reduction A $\rightarrow \alpha$ on symbols in FOLLOW(A)
  - SLR Parsing
  - Construct SLR Parsing Table
  - Moves of an SLR parser on input using SLR Parsing Table

SLR, Ambiguity, and Conflicts

- SLR grammars are unambiguous
- But **not** every unambiguous grammar is SLR
- Consider for example the unambiguous grammar

1. $S \rightarrow L = R$
2. $S \rightarrow R$
3. $L \rightarrow * R$
4. $L \rightarrow \textbf{id}$
5. $R \rightarrow L$

I_0:
$S' \rightarrow \bullet S$
$S \rightarrow \bullet L{=}R$
$S \rightarrow \bullet R$
$L \rightarrow \bullet {*}R$
$L \rightarrow \bullet \textbf{id}$
$R \rightarrow \bullet L$

I_1:
$S' \rightarrow S \bullet$

I_2:
$S \rightarrow L \bullet {=} R$
$R \rightarrow L \bullet$

I_3:
$S \rightarrow R \bullet$

I_4:
$L \rightarrow * \bullet R$
$R \rightarrow \bullet L$
$L \rightarrow \bullet {*}R$
$L \rightarrow \bullet \textbf{id}$

I_5:
$L \rightarrow \textbf{id} \bullet$

I_6:
$S \rightarrow L{=}\bullet R$
$R \rightarrow \bullet L$
$L \rightarrow \bullet {*}R$
$L \rightarrow \bullet \textbf{id}$

I_7:
$L \rightarrow {*}R \bullet$

I_8:
$R \rightarrow L \bullet$

I_9:
$S \rightarrow L{=}R \bullet$

$action[2,=]=s6$
$action[2,=]=r5$

no

Conflict: has no SLR parsing table!

Viable Prefixes

- During the LR parsing, the stack contents must be a prefix of a right-sentential form
  - If the stack holds α, the rest of input is x
  - There is a right-most derivation $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha x$
- But, not all prefixes of right-sentential forms can appear on the stack
  - The parser must not shift past the handle
  - Example: Suppose $E \underset{rm}{\overset{*}{\Rightarrow}} F * \mathbf{id} \underset{rm}{\Rightarrow} (E) * \mathbf{id}$

    the stack must not hold $(E)*$, as (E) is a handle.
- The prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes

Viable Prefixes (Cont.)

- A viable prefix is a prefix of a right-sentential form that does not continue past the right end of the leftmost handle of that sentential form

- We say item $A \rightarrow \beta_1 \bullet \beta_2$ is valid for a viable prefix $\alpha\beta_1$ if there is a derivation $S' \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta_1 \bullet \beta_2\, w$.

- $A \rightarrow \beta_1 \bullet \beta_2$ is valid for $\alpha\beta_1$ and $\alpha\beta_1$ is on the parsing stack
  - If $\beta_2 \neq \varepsilon$, then shift
  - $\beta_2 = \varepsilon$, then reduce

Viable Prefixes (Cont.)

- The set of valid items for a viable prefix δ is exactly the set of items reached from the initial state along the path labeled δ in the LR(0) automaton for the grammar
- Example: See state 7 of automaton on next slide.

  $T \to \underbrace{T*}_{\beta_1}\bullet\underbrace{F}_{\beta_2}$, $F \to \bullet\underbrace{(E)}_{\beta_2}$, and $F \to \bullet\underbrace{\textbf{id}}_{\beta_2}$ are valid items for viable prefix E+T$*$

$$
\begin{aligned}
E' &\underset{rm}{\Rightarrow} E \\
&\underset{rm}{\Rightarrow} E + T \\
&\underset{rm}{\Rightarrow} \underbrace{E +}_{\alpha} \underbrace{T *}_{\beta_1} \underbrace{F}_{\beta_2}
\end{aligned}
\qquad
\begin{aligned}
E' &\underset{rm}{\Rightarrow} E \\
&\underset{rm}{\Rightarrow} E + T \\
&\underset{rm}{\Rightarrow} E + T * F \\
&\underset{rm}{\Rightarrow} \underbrace{E + T *}_{\alpha} \underbrace{(E)}_{\beta_2}
\end{aligned}
\qquad
\begin{aligned}
E' &\underset{rm}{\Rightarrow} E \\
&\underset{rm}{\Rightarrow} E + T \\
&\underset{rm}{\Rightarrow} E + T * F \\
&\underset{rm}{\Rightarrow} \underbrace{E + T *}_{\alpha} \underbrace{\textbf{id}}_{\beta_2}
\end{aligned}
$$

LR(0) Automaton for expression

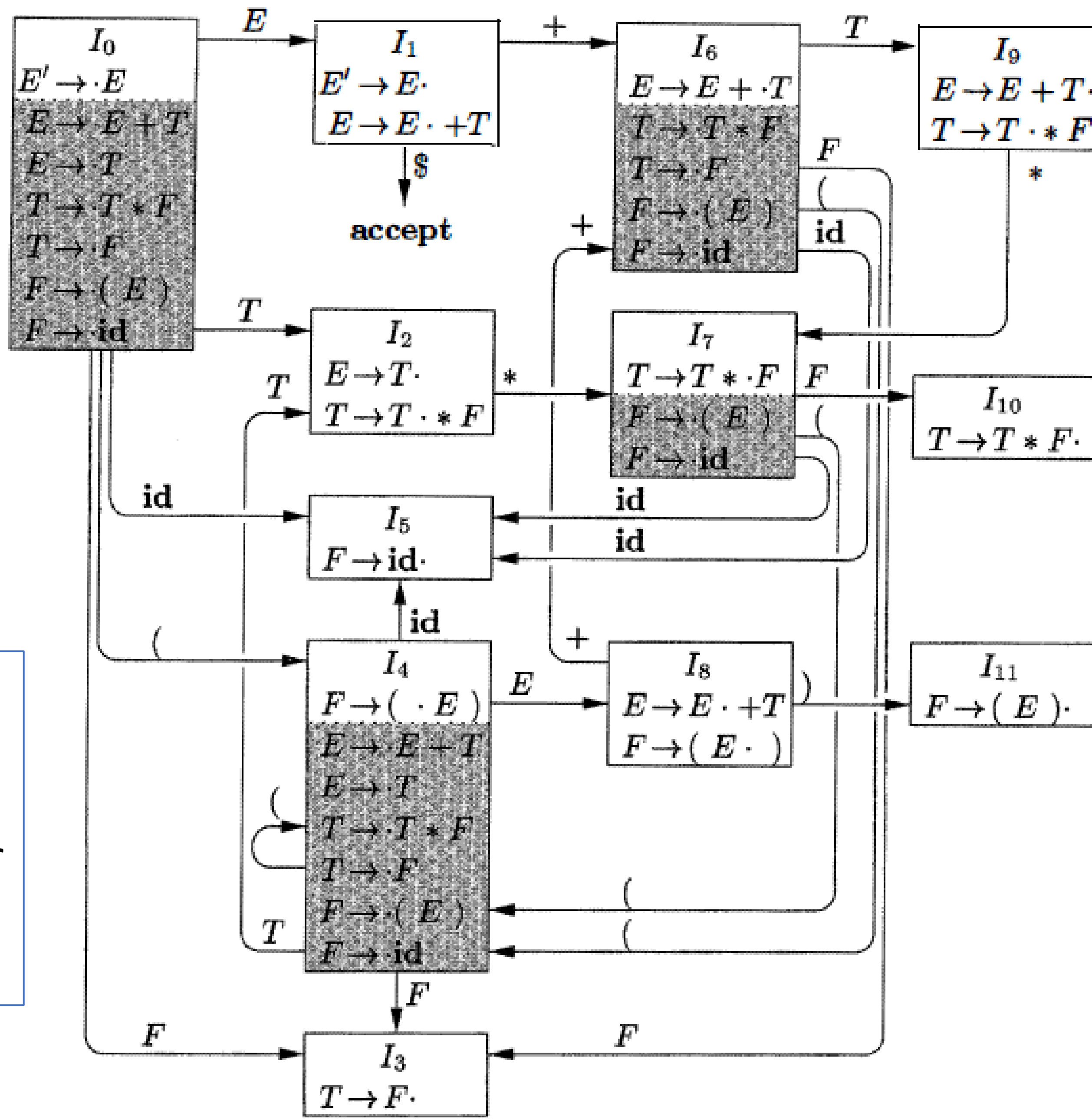
Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E)$

$F \rightarrow \mathbf{id}$



$T \rightarrow T * \bullet F$,

$F \rightarrow \bullet (E)$, and

$F \rightarrow \bullet \mathbf{id}$

are valid items for viable prefix

E+T*

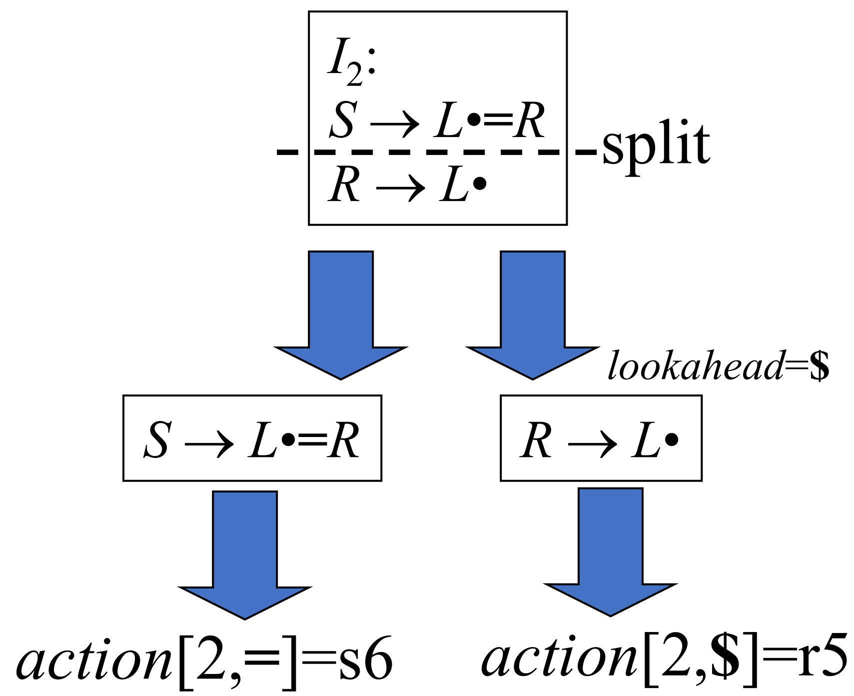
7. LR(1) Grammars

- SLR too simple
- LR(1) parsing uses lookahead to avoid unnecessary conflicts in parsing table
- LR(1) item = LR(0) item + lookahead

LR(0) item:	LR(1) item:
$[A \to \alpha \bullet \beta]$	$[A \to \alpha \bullet \beta, a]$

SLR Versus LR(1)

- Split the SLR states by adding LR(1) lookahead
- Unambiguous grammar
  1. $S \to L = R$
  2. $S \to R$
  3. $L \to * \ R$
  4. $L \to \textbf{id}$
  5. $R \to L$



Should not reduce on =, because no right-sentential form begins with $R=$

LR(1) Items

- An *LR*(1) *item*

  $[A \to \alpha \bullet \beta,\ a]$

  contains a *lookahead* terminal *a*, meaning α already on top of the stack, expect to parse βa
- For items of the form

  $[A \to \alpha \bullet,\ a]$

  the lookahead *a* is used to reduce $A \to \alpha$ only if the next lookahead of the input is *a*
- For items of the form

  $[A \to \alpha \bullet \beta,\ a]$

  with $\beta \neq \varepsilon$ the lookahead has no effect

The Closure Operation for LR(1) Items

1. Start with *closure*(I) = I
2. If $[A \to \alpha \bullet B\beta, a] \in$ *closure*(I) then for each production $B \to \gamma$ in the grammar and each terminal $b \in \text{FIRST}(\beta a)$, add the item $[B \to \bullet\gamma, b]$ to *closure*(I) if not already in *closure*(I)
3. Repeat 2 until no new items can be added

The Goto Operation for LR(1) Items

1. For each item $[A \rightarrow \alpha \bullet X\beta,\ a] \in I$, add the set of items *closure*($\{[A \rightarrow \alpha X \bullet \beta,\ a]\}$) to *goto*($I$,$X$) if not already there
2. Repeat step 1 until no more items can be added to *goto*(I,X)

Constructing the set of LR(1) Items of a Grammar

1. Augment the grammar with a new start symbol S' and production $S' \rightarrow S$
2. Initially, set C = { *closure*({[$S' \rightarrow \bullet S$, **$**]}) }
   (this is the start state of the DFA)
3. For each set of items $I \in C$ and each grammar symbol $X \in (N \cup T)$ such that *goto*(I,X) $\notin C$ and *goto*(I,X) $\neq \emptyset$, add the set of items *goto*(I,X) to C
4. Repeat 3 until no more sets can be added to C

Example Grammar and LR(1) Items

- Augmented LR(1) grammar (4.55):

  $S' \rightarrow S$
  $S \rightarrow C\ C$
  $C \rightarrow$ c C | d

- LR(1) items

$$I_0: \quad \begin{array}{l} S \rightarrow \cdot S, \$ \\ S \rightarrow \cdot CC,\ \$ \\ C \rightarrow \cdot cC,\ c/d \\ C \rightarrow \cdot d,\ c/d \end{array}$$

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

$$I_2: \quad \begin{array}{l} S \rightarrow C\cdot C,\ \$ \\ C \rightarrow \cdot cC,\ \$ \\ C \rightarrow \cdot d,\ \$ \end{array}$$

$$I_3: \quad \begin{array}{l} C \rightarrow c\cdot C,\ c/d \\ C \rightarrow \cdot cC,\ c/d \\ C \rightarrow \cdot d,\ c/d \end{array}$$

$$I_4: \quad C \rightarrow d\cdot,\ c/d$$

$$I_5: \quad S \rightarrow CC\cdot, \$$$

$$I_6: \quad \begin{array}{l} C \rightarrow c\cdot C,\ \$ \\ C \rightarrow \cdot cC,\ \$ \\ C \rightarrow \cdot d,\ \$ \end{array}$$

$$I_7: \quad C \rightarrow d\cdot,\ \$$$

$$I_8: \quad C \rightarrow cC\cdot,\ c/d$$

$$I_9: \quad C \rightarrow cC\cdot,\ \$$$

LR(1) items and goto Operation for Grammar (4.55)

$S' \to S$
$S \to C\,C$
$C \to c\,C \mid d$

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

goto(I_0, S) = I_1
goto(I_0, C) = I_2
goto(I_0, c) = I_3
goto(I_0, d) = I_4

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

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

goto(I_2, C) = I_5
goto(I_2, c) = I_6
goto(I_2, d) = I_7

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

goto(I_3, C) = I_8
goto(I_3, c) = I_3
goto(I_3, d) = I_4

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

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

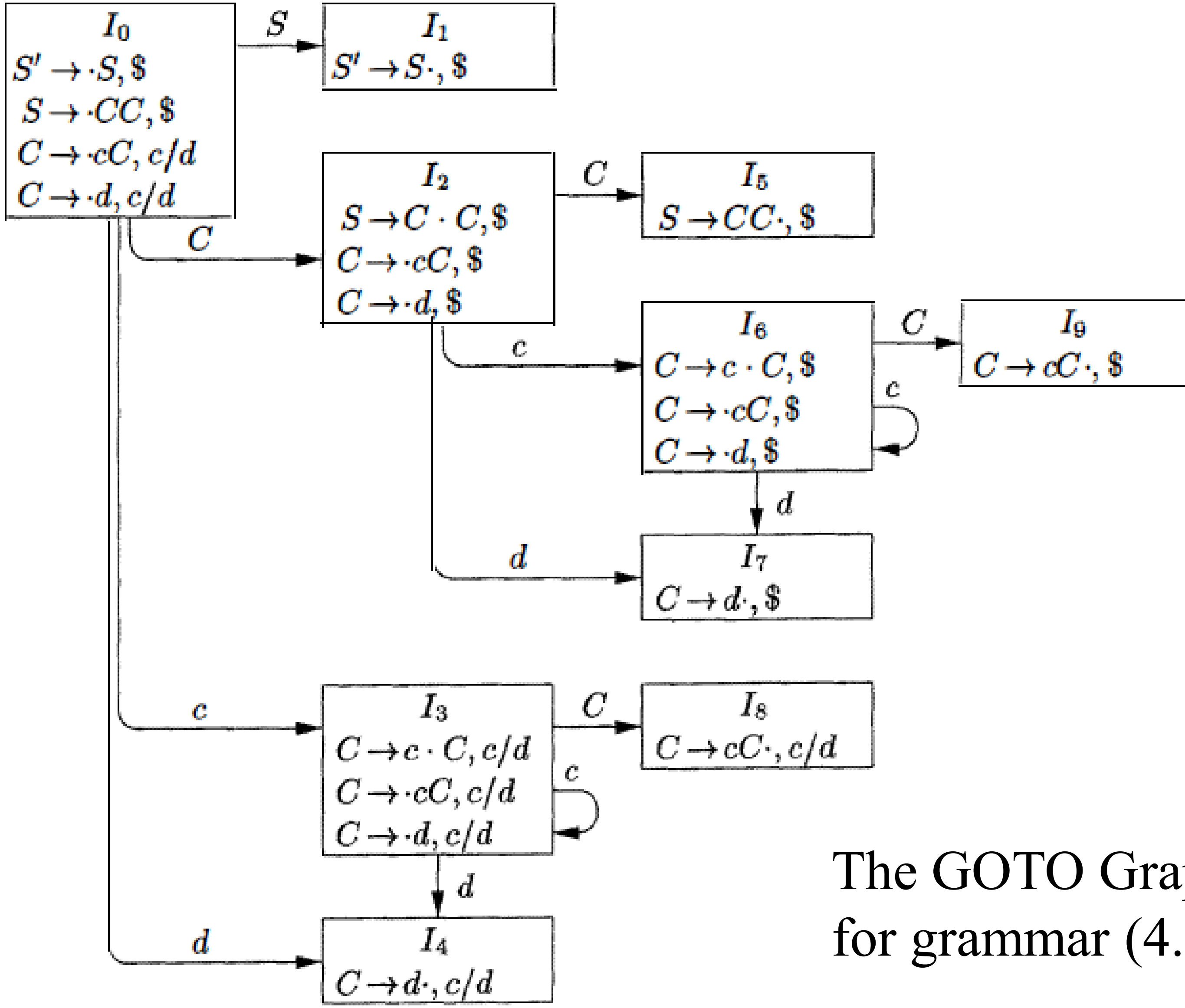
$I_6:$ $C \to c\cdot C,\ \$$
$C \to \cdot cC,\ \$$
$C \to \cdot d,\ \$$

goto(I_6, C) = I_9
goto(I_6, c) = I_6
goto(I_6, d) = I_7

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

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

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



The GOTO Graph for grammar (4.55)

Example Grammar and LR(1) Items

- Unambiguous LR(1) grammar:

  $S \rightarrow L$ **=** R
  $S \rightarrow R$
  $L \rightarrow$ ***** R
  $L \rightarrow$ **id**
  $R \rightarrow L$
- Augment with $S' \rightarrow S$
- LR(1) items (next slide)

I_0: $[S' \to \bullet S,\ \$]$ goto(I_0,S)=I_1
$[S \to \bullet L{=}R,\ \$]$ goto(I_0,L)=I_2
$[S \to \bullet R,\ \$]$ goto(I_0,R)=I_3
$[L \to \bullet {*}R,\ =]$ goto(I_0,*)=I_4
$[L \to \bullet \mathbf{id},\ =]$ goto(I_0,**id**)=I_5
$[R \to \bullet L,\ \$]$

I_1: $[S' \to S\bullet,\ \$]$

I_2: $[S \to L\bullet{=}R,\ \$]$ goto(I_2,=)=I_6
$[R \to L\bullet,\ \$]$

I_3: $[S \to R\bullet,\ \$]$

I_4: $[L \to {*}\bullet R,\ =]$ goto(I_4,R)=I_7
$[R \to \bullet L,\ =]$ goto(I_4,L)=I_8
$[L \to \bullet {*}R,\ =]$ goto(I_4,*)=I_4
$[L \to \bullet \mathbf{id},\ =]$ goto(I_4,**id**)=I_5

I_5: $[L \to \mathbf{id}\bullet,\ =]$

I_6: $[S \to L{=}\bullet R,\ \$]$ goto(I_6,R)=I_9
$[R \to \bullet L,\ \$]$ goto(I_6,L)=I_{10}
$[L \to \bullet {*}R,\ \$]$ goto(I_6,*)=I_{11}
$[L \to \bullet \mathbf{id},\ \$]$ goto(I_6,**id**)=I_{12}

I_7: $[L \to {*}R\bullet,\ =]$

I_8: $[R \to L\bullet,\ =]$

I_9: $[S \to L{=}R\bullet,\ \$]$

I_{10}: $[R \to L\bullet,\ \$]$

I_{11}: $[L \to {*}\bullet R,\ \$]$ goto(I_{11},R)=I_{13}
$[R \to \bullet L,\ \$]$ goto(I_{11},L)=I_{10}
$[L \to \bullet {*}R,\ \$]$ goto(I_{11},*)=I_{11}
$[L \to \bullet \mathbf{id},\ \$]$ goto(I_{11},**id**)=I_{12}

I_{12}: $[L \to \mathbf{id}\bullet,\ \$]$

I_{13}: $[L \to {*}R\bullet,\ \$]$

Grammar
$S \to L = R$
$S \to R$
$L \to {*}\ R$
$L \to \mathbf{id}$
$R \to L$

The GOTO Graph for Grammar

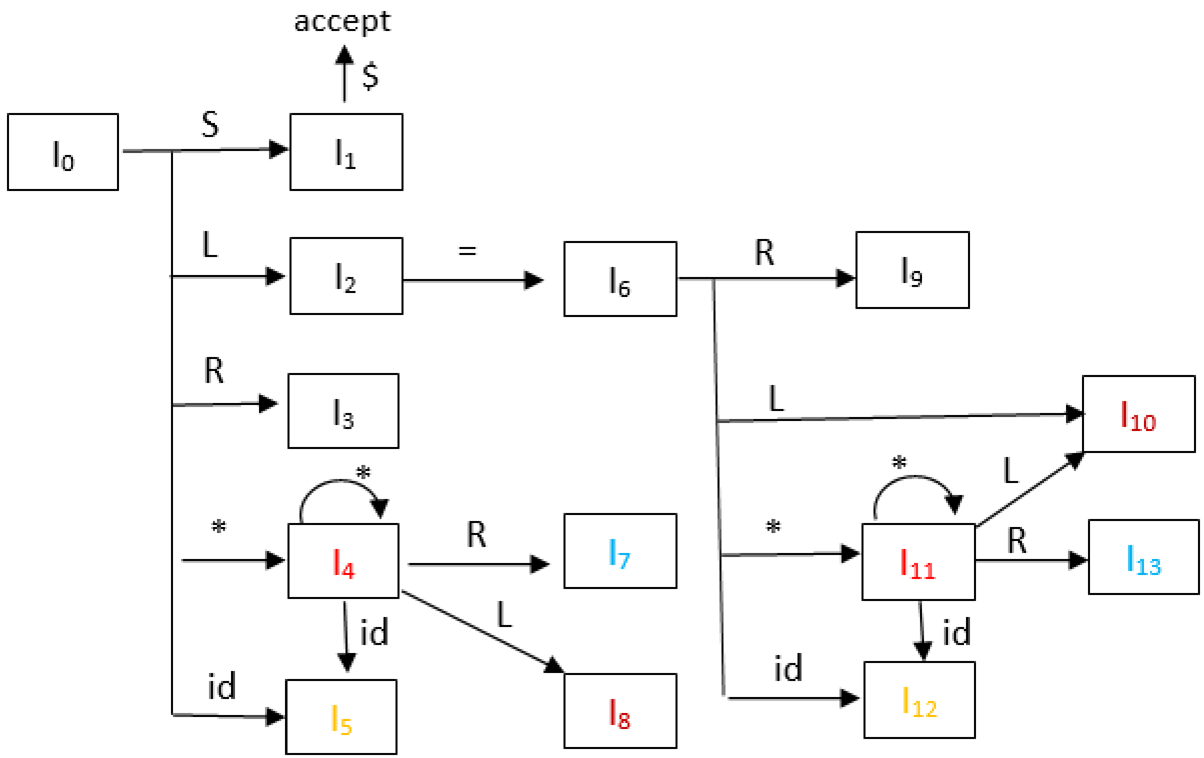
S → *L* **=** *R*

S → *R*

L → ***** *R*

L → **id**

R → *L*

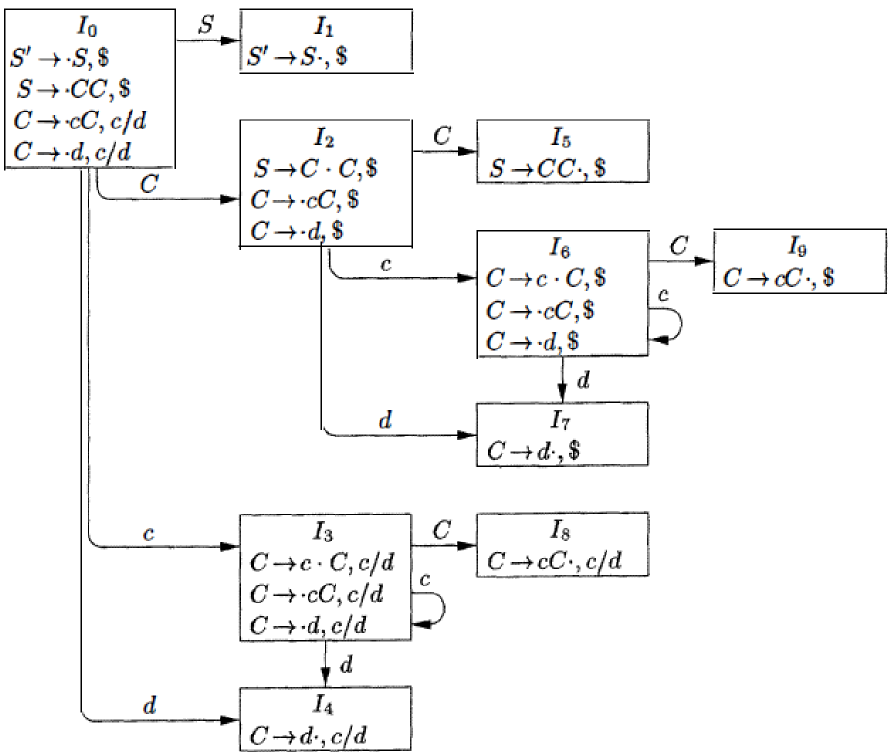


Constructing Canonical LR(1) Parsing Tables

1. Augment the grammar with $S' \rightarrow S$
2. Construct the set $C=\{I_0,I_1,\ldots,I_n\}$ of LR(1) items
3. If $[A \rightarrow \alpha \bullet a\beta, b] \in I_i$ and $goto(I_i,a)=I_j$ then set $action[i,a]$=shift j
4. If $[A \rightarrow \alpha \bullet, a] \in I_i$ then set $action[i,a]$=reduce $A \rightarrow \alpha$ (apply only if $A \neq S'$)
5. If $[S' \rightarrow S \bullet, \mathbf{\$}]$ is in I_i then set $action[i,\mathbf{\$}]$=accept
6. If $goto(I_i,A)=I_j$ then set $goto[i,A]=j$
7. Repeat 3-6 until no more entries added
8. The initial state i is the I_i holding item $[S' \rightarrow \bullet S,\mathbf{\$}]$

Example Canonical LR(1) Parsing Table

Grammar:
0. $S' \rightarrow S$
1. $S \rightarrow C\ C$
2. $C \rightarrow \boldsymbol{c}\ C$
3. $C \rightarrow \boldsymbol{d}$



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		

Example LR(1) Parsing Table

Grammar:

1. $S' \to S$
2. $S \to L = R$
3. $S \to R$
4. $L \to * R$
5. $L \to \textbf{id}$
6. $R \to L$

	id	*	=	$	*S*	*L*	*R*
0	s5	s4			1	2	3
1				acc			
2			s6	r6			
3				r3			
4	s5	s4				8	7
5			r5	r5			
6	s12	s11				10	9
7			r4	r4			
8			r6	r6			
9				r2			
10				r6			
11	s12	s11				10	13
12				r5			
13				r4			

LALR Parsing

- LR(1) parsing tables have many states
- LALR parsing (Look-Ahead LR) merges two or more LR(1) state into one state to reduce table size
- Less powerful than LR(1)
  - Will not introduce shift-reduce conflicts, because shifts do not use lookaheads
  - May introduce reduce-reduce conflicts, but seldom do so for grammars of programming languages