Syntax Analysis [Chapter 4]

Lectures 11

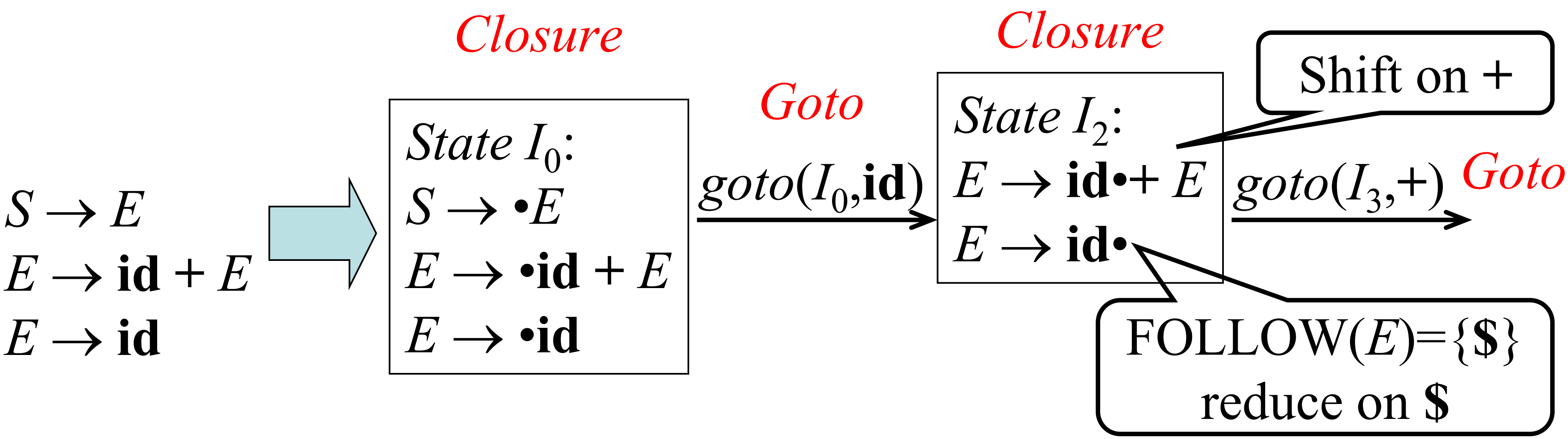
Edited by Dr. Ismail Hababeh
German Jordanian University
Adapted from slides by Dr. Robert A. Engelen

SLR Grammars

- SLR (Simple LR): a simple extension of LR(0) automation that used to eliminate some LR(0) conflicts.
- LR(0) automation use 0 lookahead (only considers the current character without reading the next character).
- SLR extend LR(0) by using Follow(A).
- SLR is a kind of lookahead (but not LR(1)).

SLR Grammars

- SLR eliminates some conflicts by populating the parsing table with reductions $A \rightarrow \alpha$ on symbols in FOLLOW(A). (it should know follow for all non-terminals). Example:



Constructing SLR Parsing Table

- LR(0) state is a set of LR(0) items
- LR(0) item is a production with a • (dot) in the right-hand side

1. Build the LR(0) DFA by constructing
   - *Closure operation* to construct LR(0) items
   - *Goto operation* to determine transitions
2. Construct the SLR parsing table from the LR(0) DFA
3. LR parser program uses the SLR parsing table to determine shift/reduce operations.

Constructing SLR Parse Table

1. Extend the grammar with $S' \to S$
2. Construct the closure set $C=\{I_0,I_1,\dots,I_n\}$ of *LR(0) items*
3. If $[A \to \alpha \bullet a \beta] \in I_i$ and $goto(I_i,a)=I_j$ then set ***action*[*i*,*a*]=shift *j*** (*a* must be a terminal).
4. If $[A \to \alpha \bullet] \in I_i$ then set ***action*[*i*,*a*]=reduce A→α** for all $a \in$ FOLLOW(A) (apply only if $A \neq S'$)
5. If $[S' \to S \bullet]$ is in I_i then set ***action*[*i*,$]=accept**
6. If $goto(I_i,A)=I_j$ then set ***goto*[*i*,*A*]=*j*** (for all nonterminals A)
7. Repeat 3-6 until no more entries added

LR(0) Grammar Items

- *LR(0) item* of a grammar G is a production of G with • at some position of the right-hand side
- Example: the following production

  $A \to X\,Y\,Z$

  has 4 items:

  $[A \to \bullet\, X\,Y\,Z]$
  $[A \to X \bullet Y\,Z]$
  $[A \to X\,Y \bullet Z]$
  $[A \to X\,Y\,Z \bullet]$

➢ Note that production $A \to \varepsilon$ has one item $[A \to \bullet]$

Steps of Constructing the Grammar Items Set

To construct the closure set $C=\{I_0,I_1,\ldots,I_n\}$ of *LR(0) items:*

1. The grammar is increased by a new start symbol S' to represent the production $S' \rightarrow S$
2. Initially, set $I_0 = closure(\{[S' \rightarrow \bullet S]\})$ (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

The Closure of LR(0) Items

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

The Goto of LR(0) Items

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

The Closure of LR(0) Items - Example

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E)$

$F \rightarrow \mathbf{id}$

new start symbol E' is created and a new production is added

$E' \rightarrow E$

$closure(\{[E' \rightarrow \bullet E]\}) =$

$\{ [E' \rightarrow \bullet E] \}$ ⇒ $\{ [E' \rightarrow \bullet E]$, $[E \rightarrow \bullet E + T]$, $[E \rightarrow \bullet T] \}$

Add $[E \rightarrow \bullet \gamma]$

⇒ $\{ [E' \rightarrow \bullet E]$, $[E \rightarrow \bullet E + T]$, $[E \rightarrow \bullet T]$, $[T \rightarrow \bullet T * F]$, $[T \rightarrow \bullet F] \}$

Add $[T \rightarrow \bullet \gamma]$

⇒ $\{ [E' \rightarrow \bullet E]$, $[E \rightarrow \bullet E + T]$, $[E \rightarrow \bullet T]$, $[T \rightarrow \bullet T * F]$, $[T \rightarrow \bullet F]$, $[F \rightarrow \bullet (E)]$, $[F \rightarrow \bullet \mathbf{id}] \}$

Add $[F \rightarrow \bullet \gamma]$

The Goto of LR(0) Items - Example 1

Grammar:

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E)$

$F \to \textbf{id}$

Assume Items of set $I = \{ [E' \to \bullet E]$
$[E \to \bullet E + T]$
$[E \to \bullet T]$
$[T \to \bullet T * F]$
$[T \to \bullet F]$
$[F \to \bullet (E)]$
$[F \to \bullet \textbf{id}] \}$

Then,

$goto(I,E) = closure(\{[E' \to E \bullet, E \to E \bullet + T]\})$

The Goto of LR(0) Items - Continue

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E)$

$F \rightarrow \mathbf{id}$

Assume Items

of set $I = \{ [E' \rightarrow E \bullet], [E \rightarrow E \bullet + T] \}$

Then, $goto(I,+) = closure(\{[E \rightarrow E + \bullet\ T]\}) = \{ [E \rightarrow E + \bullet\ T]$

$[T \rightarrow \bullet\ T * F]$

$[T \rightarrow \bullet\ F]$

$[F \rightarrow \bullet\ (E)]$

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

SLR Grammar - Example

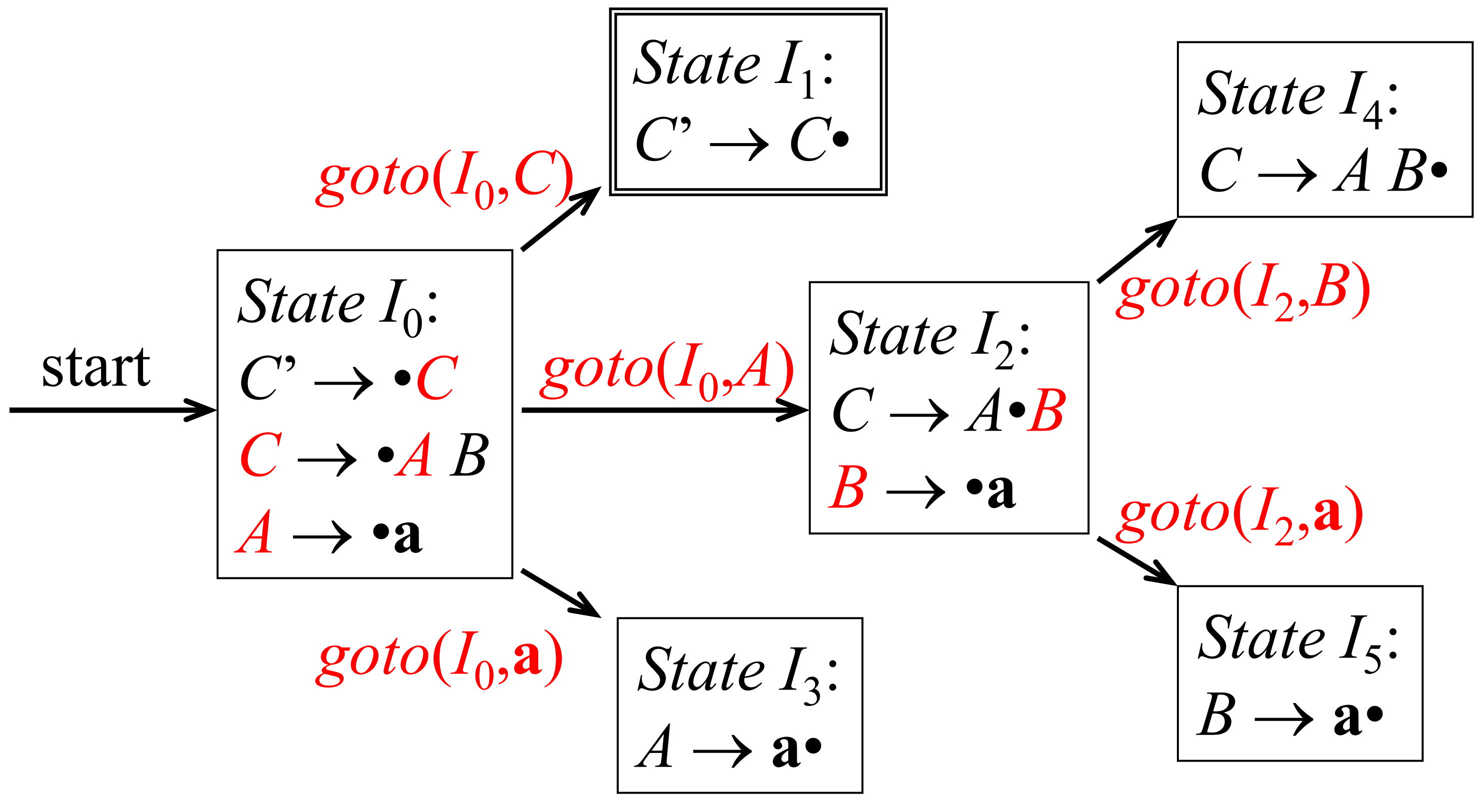
Expanded grammar:

1. $C' \to C$
2. $C \to A\ B$
3. $A \to \mathbf{a}$
4. $B \to \mathbf{a}$

$I_0 = closure(\{[C' \to \bullet C]\})$

$I_1 = \text{goto}(I_0,C) = closure(\{[C' \to C\bullet]\})$

…



Constructing SLR Parse Table

State I_0:
$C' \to \bullet C$
$C \to \bullet A\ B$
$A \to \bullet \mathbf{a}$

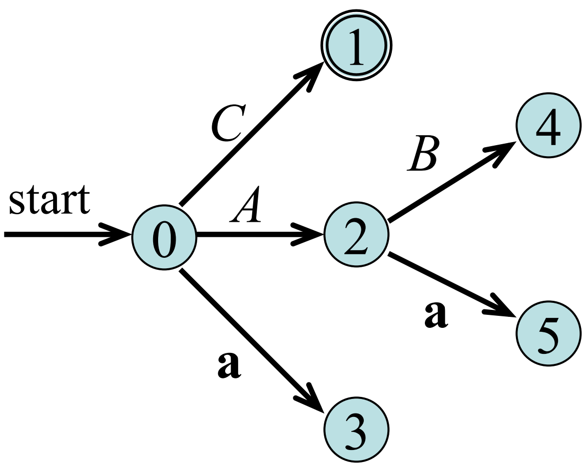
State I_1:
$C' \to C\bullet$

State I_2:
$C \to A \bullet B$
$B \to \bullet \mathbf{a}$

State I_3:
$A \to \mathbf{a}\bullet$

State I_4:
$C \to A\ B\bullet$

State I_5:
$B \to \mathbf{a}\bullet$



	a	**$**	*C*	*A*	*B*
0	s3		1	2	
1		acc			
2	s5				4
3	r3				
4		r2			
5		r4			

Expanded grammar:

1. $C' \to C$
2. $C \to A\ B$
3. $A \to \mathbf{a}$
4. $B \to \mathbf{a}$

SLR and Ambiguity

- Every SLR grammar is unambiguous, but **not** every unambiguous grammar is SLR
- Example: Consider the unambiguous grammar

$$S \rightarrow L = R \mid R$$
$$L \rightarrow * R \mid \mathbf{id}$$
$$R \rightarrow L$$

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

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:
$L \rightarrow * \bullet R$
$R \rightarrow \bullet L$
$L \rightarrow \bullet * R$
$L \rightarrow \bullet \mathbf{id}$

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

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

I_7:
$L \rightarrow * R \bullet$

I_8:
$R \rightarrow L \bullet$

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

$action[I_2, =]$ shift & goto 6
$action[I_2, =]$ reduce $(R \rightarrow L)$

The $action[I_2, =]$ could be shift/reduce entry $\Rightarrow$ (conflict).
The Grammar is not SLR. Solution is LR(1).

LR(1) Grammars

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

$[A \rightarrow \alpha \bullet \beta]$ LR(0) item

$[A \rightarrow \alpha \bullet \beta, a]$ LR(1) item

a has no effect on the item if beta is not Epsilon. But an item [A $\longrightarrow$ alpha dot, a] calls for a reduction by [A $\longrightarrow$ alpha] only if the next symbol is a.

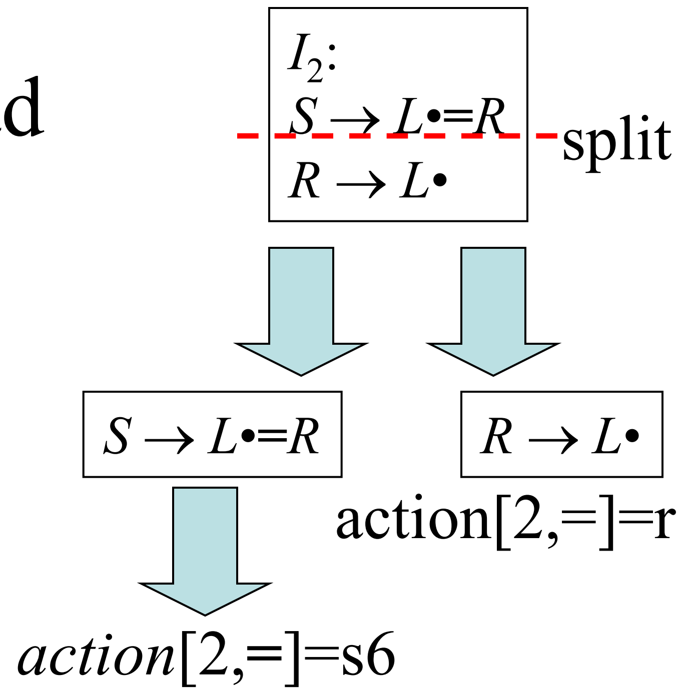
LR(1) Items Shift/Reduce

- *LR(1) item* $[A \to \alpha \bullet \beta, a]$ contains *lookahead* terminal a, meaning α already on top of the stack, expect to see βa
- For items of the form $[A \to \alpha \bullet, a]$ lookahead a is used to reduce $A \to \alpha$ only if the next input is a
- For items of the form $[A \to \alpha \bullet \beta, a]$ with $\beta \neq \varepsilon$ the lookahead has no effect

LR(1) Ambiguity Elimination

- Split the SLR states by adding LR(1) lookahead
- Grammar

$$S \rightarrow L = R \mid R$$
$$L \rightarrow * R \mid \text{id}$$
$$R \rightarrow L$$



If the next input (lookahead) is "=", don't reduce because no right-sentential form of the grammar begins with $=R$

Constructing LR(1) Items - Example

- Construct the LR(1) items for the following grammar:

  $S \rightarrow C\ C$

  $C \rightarrow \mathrm{c}\ \mathrm{C} \mid \mathrm{d}$

1. Expand the grammar with $S' \rightarrow S$
2. LR(1) items (next slide)

I_0: $[S' \to \bullet S, \$]$ goto$(I_0,S)=I_1$
$[S \to \bullet CC, \$]$ goto$(I_0,C)=I_2$
$[C \to \bullet cC, c/d]$ goto$(I_0,c)=I_3$
$[C \to \bullet d, c/d]$ goto$(I_0,d)=I_4$

LR(1) grammar:
$S \to C\ C$
$C \to c\ C \mid d$

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

I_2: $[S \to C\bullet C, \$]$ goto$(I_2,C)=I_5$
$[C \to \bullet cC, \$]$ goto$(I_2,c)=I_6$
$[C \to \bullet d, \$]$ goto$(I_2,d)=I_7$

I_3: $[C \to c\bullet C, c/d]$ goto$(I_3,d)=I_8$
$[C \to \bullet cC, c/d]$ goto$(I_3,c)=I_6$
$[C \to \bullet d, c/d]$ goto$(I_3,C)=I_7$

I_4: $[C \to d\bullet, c/d]$

I_5: $[C \to CC\bullet, \$]$

I_6: $[C \to c\bullet C, \$]$ goto$(I_6,C)=I_9$
$[C \to \bullet cC, \$]$ goto$(I_6,c)=I_6$
$[C \to \bullet d, \$]$ goto$(I_6,d)=I_7$

I_7: $[C \to d\bullet, \$]$

I_8: $[C \to cC\bullet, c/d]$

I_9: $[C \to cC\bullet, \$]$

(LookAhead LR) LALR(1) Parsing Grammars

- Its table is smaller than LR(1)
- LR(1) parsing tables have many states
- LALR(1) combines LR(1) states to reduce table size
- Less powerful than LR(1)
  - Will not introduce shift-reduce conflicts, because shifts don't use lookaheads
  - May introduce reduce-reduce conflicts, but not often for grammars of programming languages.

Constructing LALR(1) Parsing Tables

1. Construct sets of LR(1) items
2. Combine LR(1) sets with sets of items that share the same first part

$$\left.\begin{array}{l} I_4\text{: } [C \to \mathrm{d}\bullet, c/d] \\ I_7\text{: } [C \to d\bullet, \$] \end{array}\right\} \quad I_{47}\text{: } [\mathrm{C} \to d\bullet, c/d/\$]$$

Shorthand for two items in the same set

Constructing LALR(1) Parse Table - Example

- LR(1) grammar:

  $S \rightarrow C\ C$

  $C \rightarrow$ c C | d

- Augment with $S' \rightarrow S$
- LALR(1) items (next slide)

Constructing LALR(1) Parse Table

I_3: $[C \rightarrow c\bullet C, c/d]$
$[C \rightarrow \bullet cC, c/d]$
$[C \rightarrow \bullet d, c/d]$

I_6: $[C \rightarrow c\bullet C, \$]$
$[C \rightarrow \bullet cC, \$]$
$[C \rightarrow \bullet d, \$]$

I_{36}: $[C \rightarrow c\bullet C, c/d/\$]$
$[C \rightarrow \bullet cC, c/d/\$]$
$[C \rightarrow \bullet d, c/d/\$]$

I_4: $[C \rightarrow d\bullet, c/d]$

I_7: $[C \rightarrow d\bullet, \$]$

I_{47}: $[C \rightarrow d\bullet, c/d/\$]$

I_8: $[C \rightarrow cC\bullet, c/d]$

I_9: $[C \rightarrow cC\bullet, \$]$

I_{89}: $[C \rightarrow cC\bullet, c/d/\$]$

LALR(1) Parse Table

Grammar:
1. $S \rightarrow C\ C$
2. $C \rightarrow c$ C
3. $C \rightarrow d$

	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		

LL, SLR, LR, LALR Summary

- LL parse tables computed using FIRST/FOLLOW
  - Nonterminals × terminals → productions
- LR parsing tables computed using closure/goto
  - LR states × terminals → shift/reduce actions
  - LR states × nonterminals → goto state transitions
- A grammar is considered
  - LL(1) if its LL(1) parse table has no conflicts
  - LR(1) if its LR(1) parse table has no conflicts
  - SLR if its SLR parse table has no conflicts
  - LALR(1) if its LALR(1) parse table has no conflicts

LL, SLR, LR, LALR Summary

- Almost all programming languages have LR grammars
- LR is more powerful than LL.

(i.e, every LL(1) grammar is also both LALR(1) and LR(1), but not vice versa).

- LR grammar is usually easier to understand than the corresponding LL grammar.
- LR parser itself is harder to understand and to write (thus, LR parsers are built using parser generators, rather than being written by hand).