Syntax Analysis
[Chapter 4 - Part 4]

Lecture 13

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

LR(*k*) Parsing

- The most common type of bottom-up parser is based on LR(*k*) parsing, where *k* is the number of input symbols of lookahead (usually 1 or 0).
- LR(0) are grammar rules with a special dot added somewhere in the right-hand side.

LR(*k*) Parsing

- LR parsers are table driven.
- LR parser generator is usually used to construct an LR parser, because it is too much work to build an LR parser by hand.

LR(*k*) Parsers' Items

- LR parser maintains states to make shift-reduce decisions.
- Each state represents a set of items
- Each item represents a production of a grammar with a dot that indicates how much of a production we have seen at a given point in the parsing process.

LR(k) Parsers' Items - Example

Example: The production $A \rightarrow XYZ$ produces 4 items:

$$\left.\begin{array}{l} A \rightarrow .XYZ \\ A \rightarrow X.YZ \\ A \rightarrow XY.Z \\ A \rightarrow XYZ. \end{array}\right)$$

This set of items is a state, which is waiting for the reduction to A

The item $A \rightarrow X.YZ$ indicates that we have seen on the input a string derivable from X and we expect to see string(s) derivable from Y Z

Defining Transitions in DFA

- The GOTO(I,X) function is used to define the transitions in the DFA called (LR(0) automation) for a grammar, where I is a transition state and X is a grammar symbol.
- GOTO(I, X) specifies the transition from the state I under production X.

GOTO (I, X) Function Definition

- GOTO (I, X) is defined as the closure of the set of all items generated from a production

  [$A \rightarrow aX \cdot ß$] such that [$A \rightarrow a \cdot Xß$] in I.

GOTO Function - Example

Given the following grammar:

$$E \rightarrow E + T$$
$$E \rightarrow T$$
$$T \rightarrow T * F$$
$$T \rightarrow F$$
$$F \rightarrow (E)$$
$$F \rightarrow \mathbf{id}$$

If I is the set of two items {[E' → E.], [E → E. + T]}, then what are the items generated by GOTO(I, +) ?

Solution

Step 1: Compute GOTO(I, +) by examining I for items with + immediately to the right of the dot, $E' \rightarrow E\cdot$ is not such an item, but $E \rightarrow E \cdot + T$ is.

Step 2: Move the dot over the + to get $E \rightarrow E + \cdot T$ and then took the closure of this singleton set.

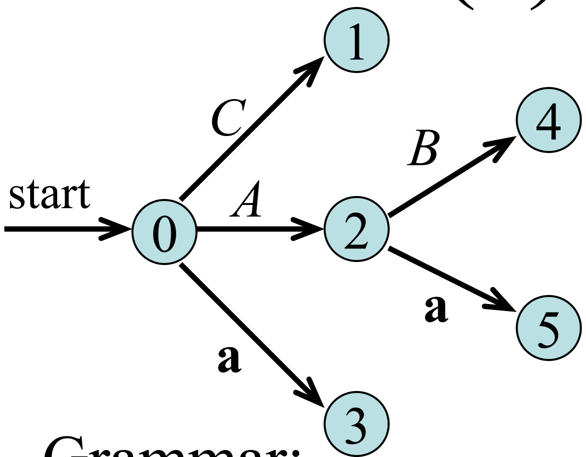
$$
\begin{aligned}
E &\rightarrow E + \cdot T \\
T &\rightarrow \cdot T * F \\
T &\rightarrow \cdot F \\
F &\rightarrow \cdot (E) \\
F &\rightarrow \cdot \mathbf{id}
\end{aligned}
$$

LR(*0*) Sets Computation

Computation of the canonical collection of sets of LR(0) items

void $items(G')$ {
    $C = \text{CLOSURE}(\{[S' \to \cdot S]\})$;
    repeat
        for (each set of items I in C)
            for (each grammar symbol X)
                if ($\text{GOTO}(I, X)$ is not empty and not in C)
                    add $\text{GOTO}(I, X)$ to C;
    until no new sets of items are added to C on a round;
}

Using DFA for Shift/Reduce LR(k) Decisions - Example



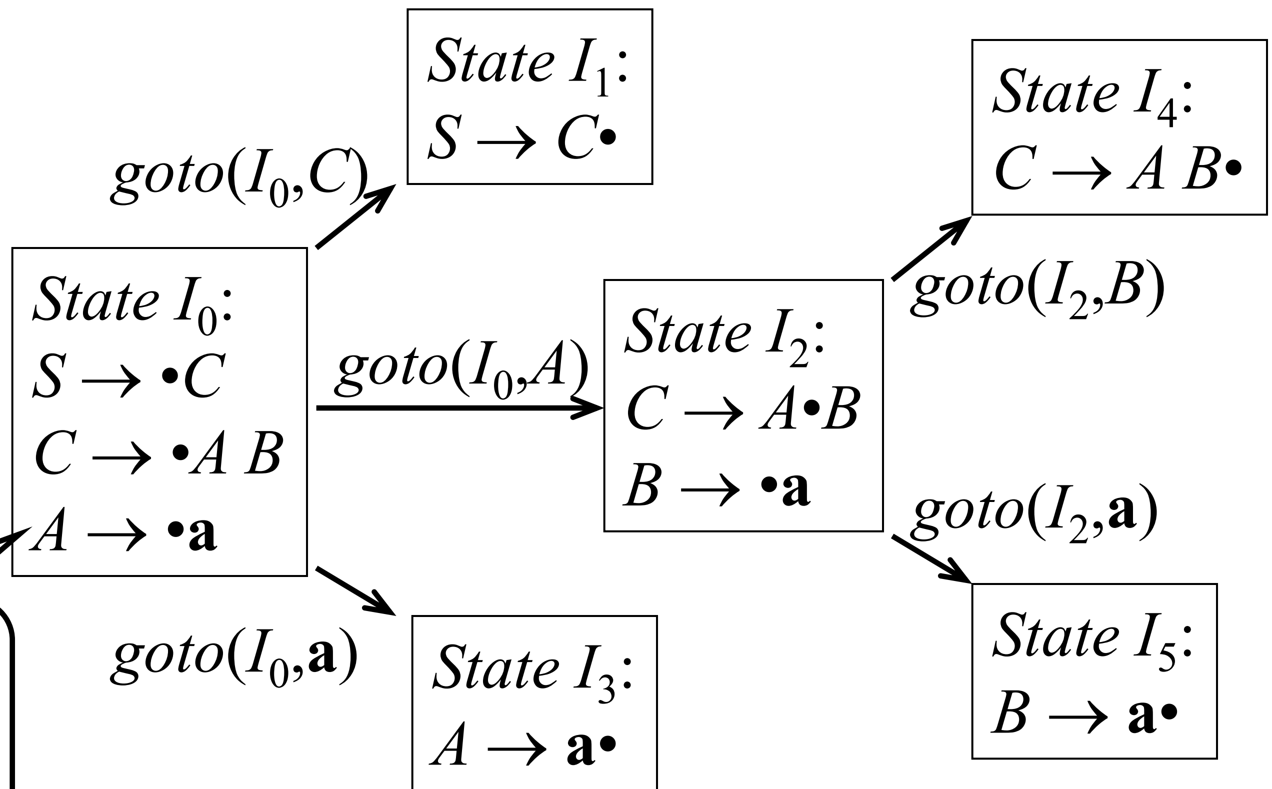
Grammar:

$S \rightarrow C$

$C \rightarrow A\ B$

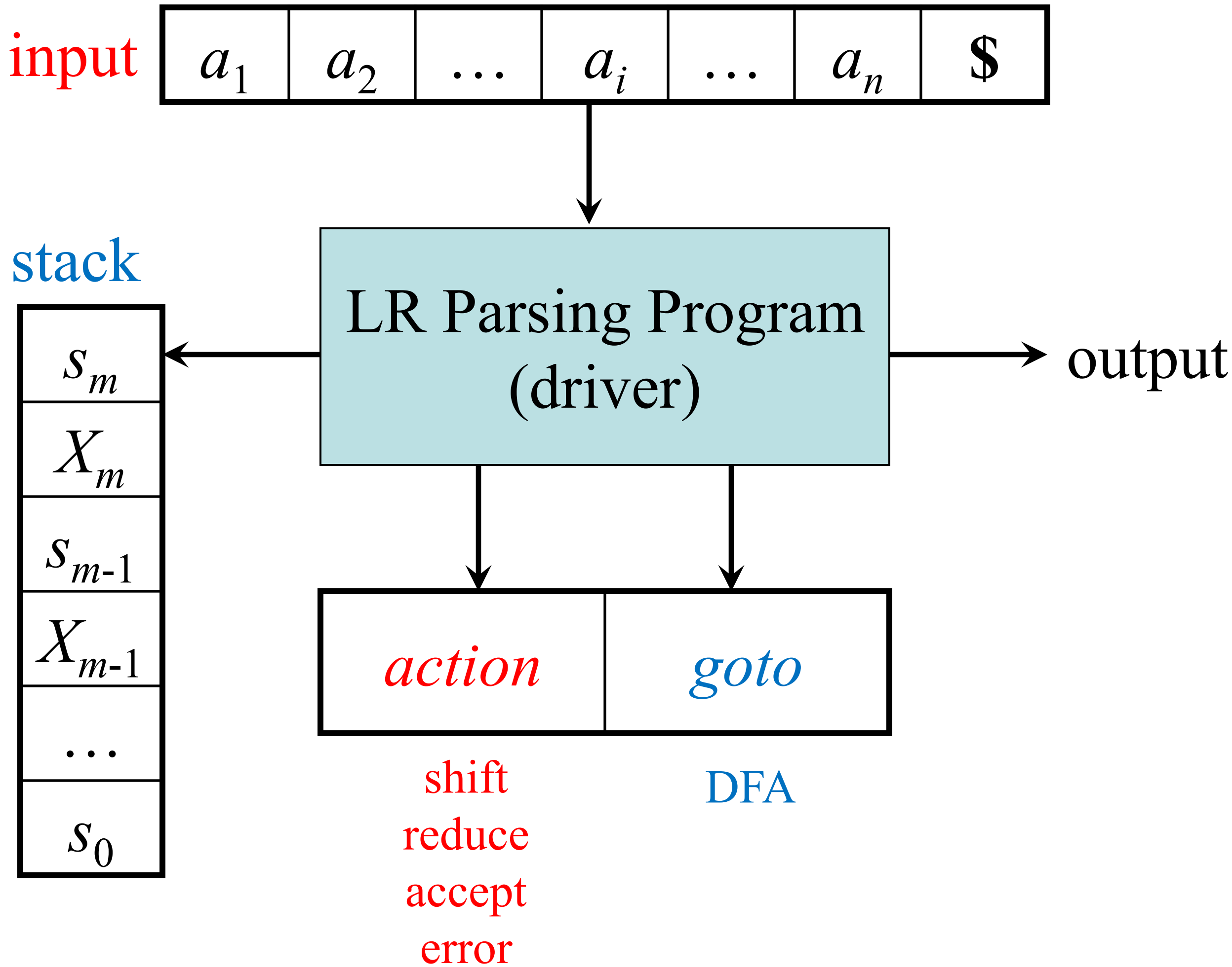
$A \rightarrow \mathbf{a}$

$B \rightarrow \mathbf{a}$



Can only reduce $A \rightarrow \mathbf{a}$ (not $B \rightarrow \mathbf{a}$)

LR Parser Implementation



All transitions to state j must be for the same grammar symbol X.
State j must be associated with grammar symbol X.

LR Parsing Mechanism

- The parser reads the current input symbol a_i and the state on the top of the stack s_m, and then consulting the entry $action[s_m,a_i]$
- The states of the DFA are used to determine if a handle is on top of the stack.

LR parser Configuration

$$(\underbrace{s_0\, X_1\, s_1\, X_2\, s_2\, \ldots\, X_m\, s_m}_{\text{stack}},\quad \underbrace{a_i\, a_{i+1}\, \ldots\, a_n\, \$}_{\text{input}})$$

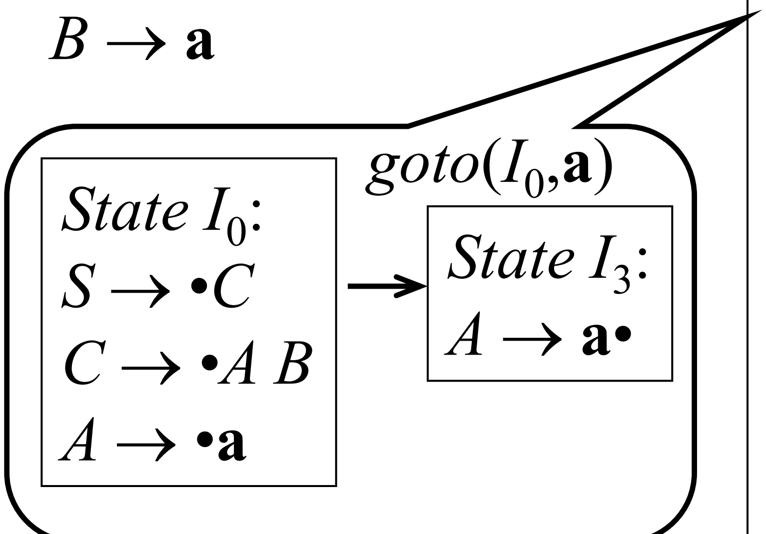
The *action* function inputs are a state s_m and a terminal a_i:

- If $action[s_m, a_i]$ = shift s, then push a_i, push s, and advance input:
  $$(s_0\, X_1\, s_1\, X_2\, s_2\, \ldots\, X_m\, s_m\, a_i\, s,\quad a_{i+1}\, \ldots\, a_n\, \$)$$
- If $action[s_m, a_i]$ = reduce $A \rightarrow \beta$ and $goto[s_{m-r}, A] = s$ with $r = |\beta|$ (length of beta) then pop 2 r symbols, push A, and push s:
  $$(s_0\, X_1\, s_1\, X_2\, s_2\, \ldots\, X_{m-r}\, s_{m-r}\, A\, s,\quad a_i\, a_{i+1}\, \ldots\, a_n\, \$)$$
- If $action[s_m, a_i]$ = accept, then stop (parsing is completed)
- If $action[s_m, a_i]$ = error, then call error recovery routine

DFA for LR Parsing - Example 1

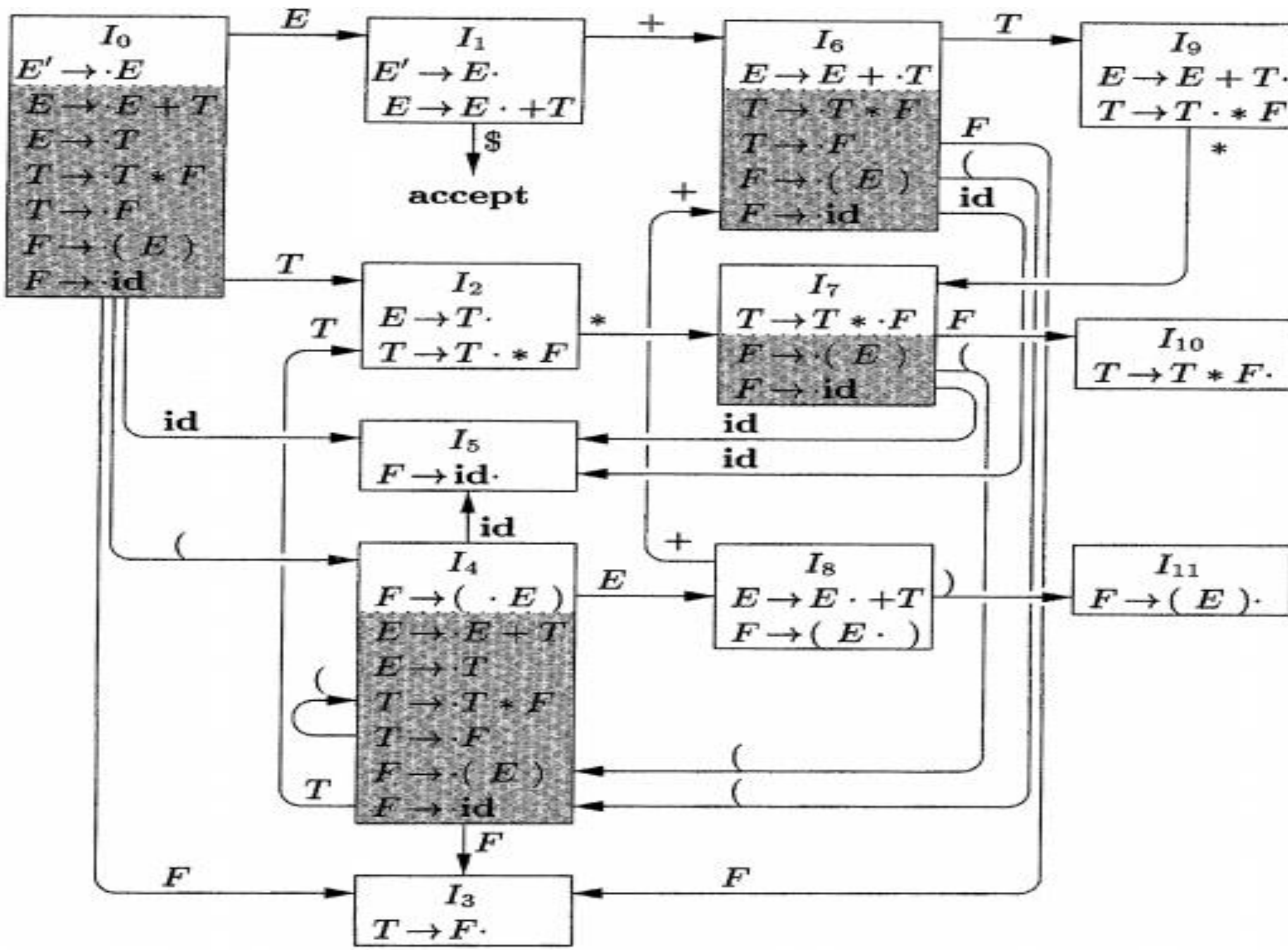
Implement the *goto(I,X)* function to check if the input *aa* is accepted by the following grammar.

$S \rightarrow C$
$C \rightarrow A\ B$
$A \rightarrow \mathbf{a}$
$B \rightarrow \mathbf{a}$



Stack	Input	Action
\$ 0	**aa\$**	start in state 0
\$ 0	**aa\$**	shift (and goto state 3)
\$ 0 $\underline{\mathbf{a}}$ 3	**a\$**	reduce $A \rightarrow \mathbf{a}$ (goto 2)
\$ 0 A 2	**a\$**	shift (and goto state 5)
\$ 0 A 2 $\underline{\mathbf{a}}$ 5	**\$**	reduce $B \rightarrow \mathbf{a}$ (goto 4)
\$ 0 $\underline{A}$ 2 $\underline{B}$ 4	**\$**	reduce $C \rightarrow AB$ (goto 1)
\$ 0 $\underline{C}$ 1	**\$**	reduce $S \rightarrow C$
\$ 0 S 1	**\$**	accept

LR Parsing – Example 2



LR Parsing (Bottom-Up) - Example

Grammar:

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow \mathbf{(}\ E\ \mathbf{)}$
6. $F \rightarrow \mathbf{id}$

Stack	Input	Action
$ 0	**id*id+id$**	shift 6
$ 0 **id** 6	***id+id$**	reduce 6 goto 3
$ 0 F 3	***id+id$**	reduce 4 goto 2
$ 0 T 2	***id+id$**	shift 7
$ 0 T 2 * 7	**id+id$**	shift 5
$ 0 T 2 * 7 **id** 5	**+id$**	reduce 6 goto 10
$ 0 T 2 * 7 F 10	**+id$**	reduce 3 goto 2
$ 0 T 2	**+id$**	reduce 2 goto 1
$ 0 E 1	**+id$**	shift 6
$ 0 E 1 + 6	**id$**	shift 5
$ 0 E 1 + 6 **id** 5	**$**	reduce 6 goto 3
$ 0 E 1 + 6 F 3	**$**	reduce 4 goto 9
$ 0 E 1 + 6 T 9	**$**	reduce 1 goto 1
$ 0 E 1	**$**	accept

LR Parse Table - Example

Grammar:

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow$ **id**

action / *goto*

input / state

	id	+	*	(	)	$	*E*	*T*	*F*
0	s6			s5			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			

Shift & goto 5

Reduce & goto 1