Chapter 5
Bottom-Up Parsing

2022 Spring&Summer

Outline

A more powerful parsing technology

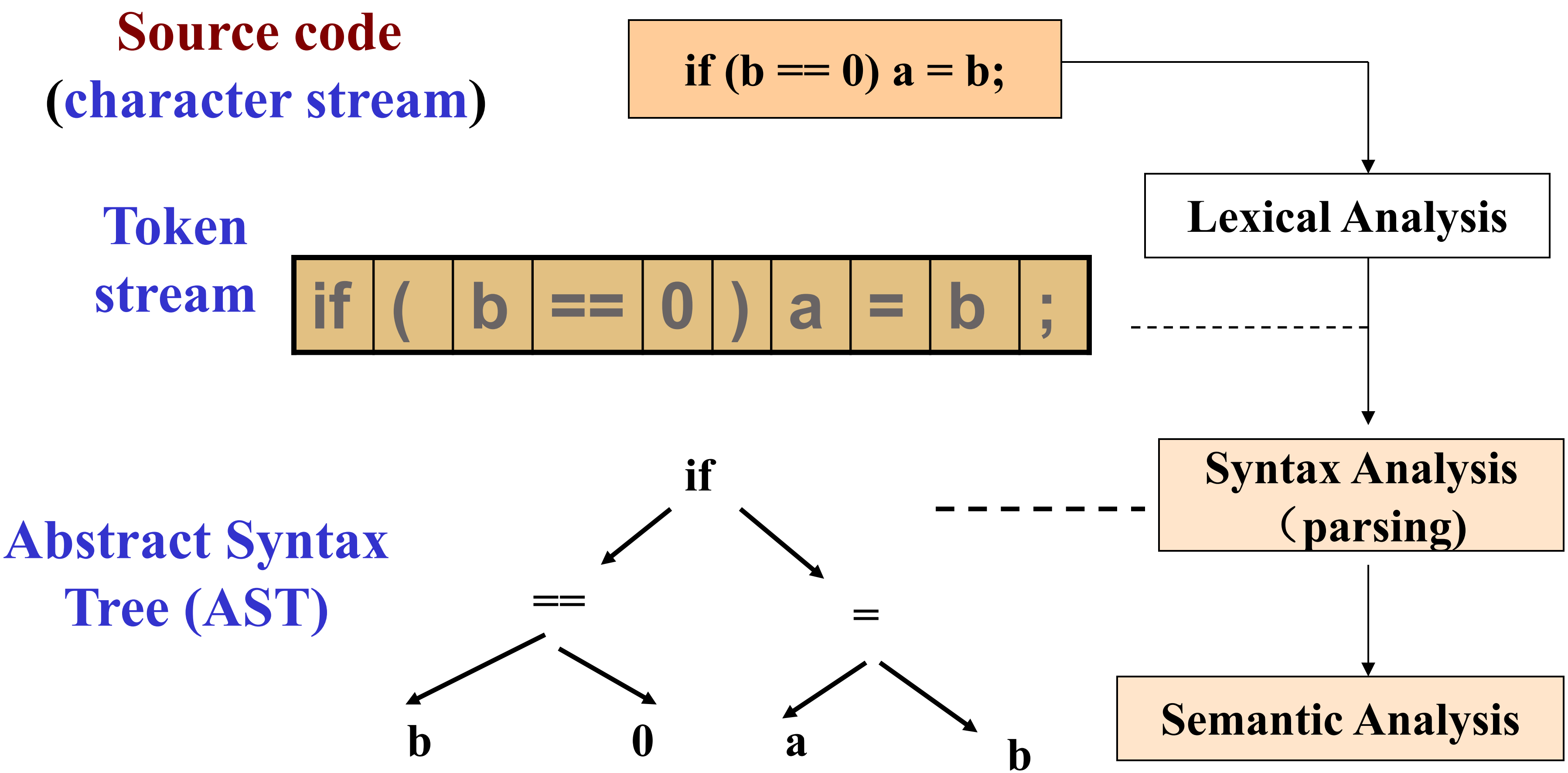
LR grammars -- more expressive than LL

- Construct right-most derivation of program
- Left-recursive grammars, virtually all programming languages
- Easier to express programming language syntax

Shift-Reduce parsers

- Parsers for LR grammars
- Automatic parser generators (e.g., YACC, CUP)

Where We Are


Source code
(character stream)
if (b == 0) a = b;
Lexical Analysis
Token stream
if (b == 0) a = b ;
Syntax Analysis
（parsing)
Abstract Syntax Tree (AST)
if
==
=
b
0
a
b
Semantic Analysis

5.1 OVERVIEW OF BOTTOM-UP PARSING

- A bottom-up parser uses an *explicit stack* to perform a parse.
- The parsing stack will contain both tokens and nonterminals.

$ *InputString* $

……….

$ *StartSymbol* $ *accept*

5.1 OVERVIEW OF BOTTOM-UP PARSING

Right-most derivation -- backward
Start with the tokens; end with the start symbol

$$S \rightarrow S + E \mid E$$
$$E \rightarrow num \mid (S)$$

(1+2+(3+4))+5 ⇐
(E+2+(3+4))+5 ⇐
(S+2+(3+4))+5 ⇐
(S+E+(3+4))+5 ⇐
(S+(3+4))+5 ⇐
(S+(E+4))+5 ⇐
(S+(S+4))+5 ⇐
(S+(S+E))+5 ⇐
(S+(S))+5 ⇐
(S+E)+5 ⇐
(S)+5 ⇐
E+5 ⇐
S+5 ⇐
S+E ⇐
S

5.1 OVERVIEW OF BOTTOM-UP PARSING

(1+2+(3+4))+5 ⇐		**(1+2+(3+4))+5**
(E+2+(3+4))+5 ⇐	(1	**+2+(3+4))+5**
(S+2+(3+4))+5 ⇐	(1	**+2+(3+4))+5**
(S+E+(3+4))+5 ⇐	(1+2	**+(3+4))+5**
(S+(3+4))+5 ⇐	(1+2+(3	**+4))+5**
(S+(E+4))+5 ⇐	(1+2+(3	**+4))+5**
(S+(S+4))+5 ⇐	(1+2+(3	**+4))+5**
(S+(S+E))+5 ⇐	(1+2+(3+4	**))+5**
(S+(S))+5 ⇐	(1+2+(3+4	**))+5**
(S+E)+5 ⇐	(1+2+(3+4)	**)+5**
(S)+5 ⇐	(1+2+(3+4)	**)+5**
E+5 ⇐	(1+2+(3+4))	**+5**
S+E ⇐	(1+2+(3+4))+5	
S	(1+2+(3+4))+5	

5.1 OVERVIEW OF BOTTOM-UP PARSING

- Parsing actions: a sequence of **shift** and **reduce** operations
- Parser state: a stack of terminals and non-terminals (grows to the right)
- Current derivation step = always stack+input

Derivation	step stack	unconsumed input
(1+2+(3+4))+5 ⇐		**(1+2+(3+4))+5**
	(	**1+2+(3+4))+5**
	(1	**+2+(3+4))+5**
(E+2+(3+4))+5 ⇐	(E	**+2+(3+4))+5**
(S+2+(3+4))+5 ⇐	(S	**+2+(3+4))+5**
	(S+	**2+(3+4))+5**
	(S+2	**+(3+4))+5**
(S+E+(3+4))+5 ⇐	(S+E	**+(3+4))+5**

5.1 OVERVIEW OF BOTTOM-UP PARSING

1. Shift : Shift a terminal from the front of the input to the top of the stack.

2. Reduce: Reduce a string α at the top of the stack to a nonterminal A, given the BNF choice $A \to \alpha$.

A bottom-up parser : a shift-reduce parser.

One further feature of bottom-up parsers： grammars are always augmented with a *new start symbol*.

if S is the start symbol, a new start symbol S' is added to the grammar : $S' \to S$

5.1 OVERVIEW OF BOTTOM-UP PARSING

- Example Consider the following augmented grammar for balanced parentheses:

  $S' \rightarrow S$

  $S \rightarrow (S)S|\varepsilon$

- Four steps of the rightmost derivation

  $$S' \Rightarrow S \Rightarrow (S)S \Rightarrow (S) \Rightarrow (\,)$$

	Parsing stack	Input	Action
1	\$	( ) \$	Shift
2	\$ (	) \$	Reduce $S \rightarrow \varepsilon$
3	\$ (S	) \$	Shift
4	\$ (S)	\$	Reduce $S \rightarrow \varepsilon$
5	\$ (S) S	\$	Reduce $S \rightarrow (S)\,S$
6	\$S	\$	Reduce $S' \rightarrow S$
7	\$S′	\$	Accept

5.1 OVERVIEW OF BOTTOM-UP PARSING

- Example. Consider the following augmented grammar for arithmetic expressions (no parentheses and one operation):

  $E' \to E \qquad E \to E + n \mid n$

- A bottom-up parse of the string $n + n$

  The *rightmost* derivation $E' \Rightarrow E \Rightarrow E+n \Rightarrow n+n$

	Parsing stack	Input	Action
1	\$	n+n\$	Shift
2	\$n	+n\$	Reduce $E \to n$
3	\$E	+n\$	Shift
4	\$E+	n\$	Shift
5	\$E+n	\$	Reduce $E \to E+n$
6	\$E	\$	Reduce $E' \to E$
7	\$E′	\$	Accept

5.1 OVERVIEW OF BOTTOM-UP PARSING

- Right Sentential Form:
  - Each of the intermediate strings of terminals and nonterminals in such a derivation is called a right sentential form.
  - Each such sentential form is split between the parsing stack and the input during a shift-reduce parse.

5.1 OVERVIEW OF BOTTOM-UP PARSING

- Right sentential form:
  - ✓ E, $E+$,and $E+n$ are all ***viable prefixes*** of the right sentential form $E+n$.
    - ✓ The right sentential form $n+n$ has ε and n as its viable prefix.
    - ✓ That $\boldsymbol{n}+$ is not a viable prefix of $n+n$.
- The sequence of symbols on the parsing stack is called ***a viable prefix*** of the right sentential form .

5.1 OVERVIEW OF BOTTOM-UP PARSING

- Handle:

This string, together with the *position* in the right sentential form where it occurs, and the production used to reduce it, is called the handle of the right sentential form.

- *Determining the next handle in a parse is the main task of a shift-reduce parser.*

- Example, in step 3 of Table 5.1 a reduction *by* $S \rightarrow \varepsilon$ could be performed

  The resulting string *(SS)* is not a right sentential form,

  thus ε is not the handle at this position in the sentential form *(S* .

5.2 FINITE AUTOMATA OF LR(0) ITEMS AND LR(0) PARSING

- An LR(0) item of a context-free grammar: a production choice with a distinguished position in its right-hand side.
- Distinguished position by a period
- If $A \rightarrow \alpha$, $\beta\gamma = \alpha$, then $A \rightarrow \beta \cdot \gamma$ is an LR(0) item.

5.2.1 LR(0) Items

- Example

$$S' \rightarrow S$$
$$S \rightarrow (S)S \mid \varepsilon$$

This grammar has three production choices and eight items:

$$S' \rightarrow \cdot S$$
$$S' \rightarrow S\cdot$$
$$S \rightarrow \cdot(S)S$$
$$S \rightarrow (\cdot S)S$$
$$S \rightarrow (S\cdot)S$$
$$S \rightarrow (S)\cdot S$$
$$S \rightarrow (S)S\cdot$$
$$S \rightarrow \cdot$$

5.2.1 LR(0) Items

- Example 5.4 The grammar of Example 5.2 has the following eight items:

  $E' \rightarrow \cdot E$

  $E' \rightarrow E\cdot$

  $E \rightarrow \cdot E + n$

  $E \rightarrow E \cdot + n$

  $E \rightarrow E + \cdot\, n$

  $E \rightarrow E + n\cdot$

  $E \rightarrow \cdot n$

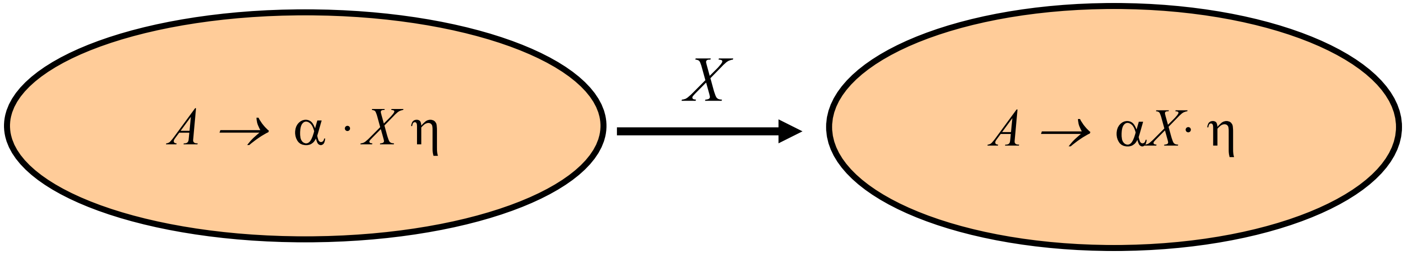
  $E \rightarrow n\cdot$

5.2.2 Finite Automata of Items

- The LR(0) items： as the states of a finite automaton
- This will start out as a nondeterministic finite automaton.
- Construct the DFA of sets of LR(0) items using the subset construction from this NFA of LR(0) items .
- Construct the DFA of sets of LR(0) items directly.

5.2.2 Finite Automata of Items

- If X is a token or a nonterminal
  the item can be written as $A \rightarrow \alpha.X\eta$



5.2.2 Finite Automata of Items

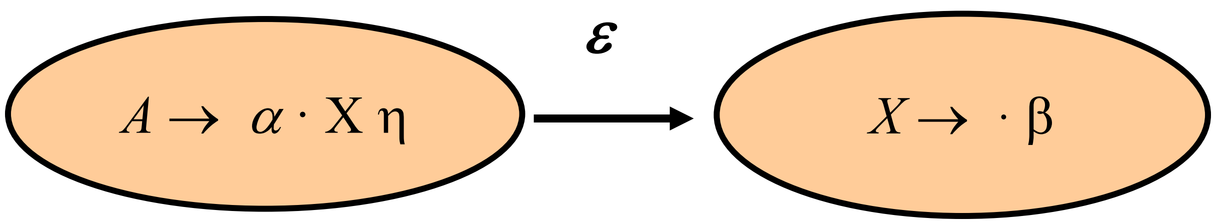
- If X is a token or a nonterminal
  the item can be written as $A \to \alpha.X\eta$

$$\boxed{A \to \alpha \cdot X\eta} \xrightarrow{X} \boxed{A \to \alpha X \cdot \eta}$$

- If X is a token, then this transition corresponds to a shift of X from the input to the top of the stack during a parse.

5.2.2 Finite Automata of Items

- If X is a nonterminal
- X will never appear as an input symbol. (such a transition will still correspond to the pushing of X onto the stack during a parse, but this can only occur during a reduction by a production $X \to \beta$.)



5.2.2 Finite Automata of Items

- The *start state* of the NFA ⟷ the *initial state* of the parser: the stack is empty

  S : the *start symbol* of the grammar.

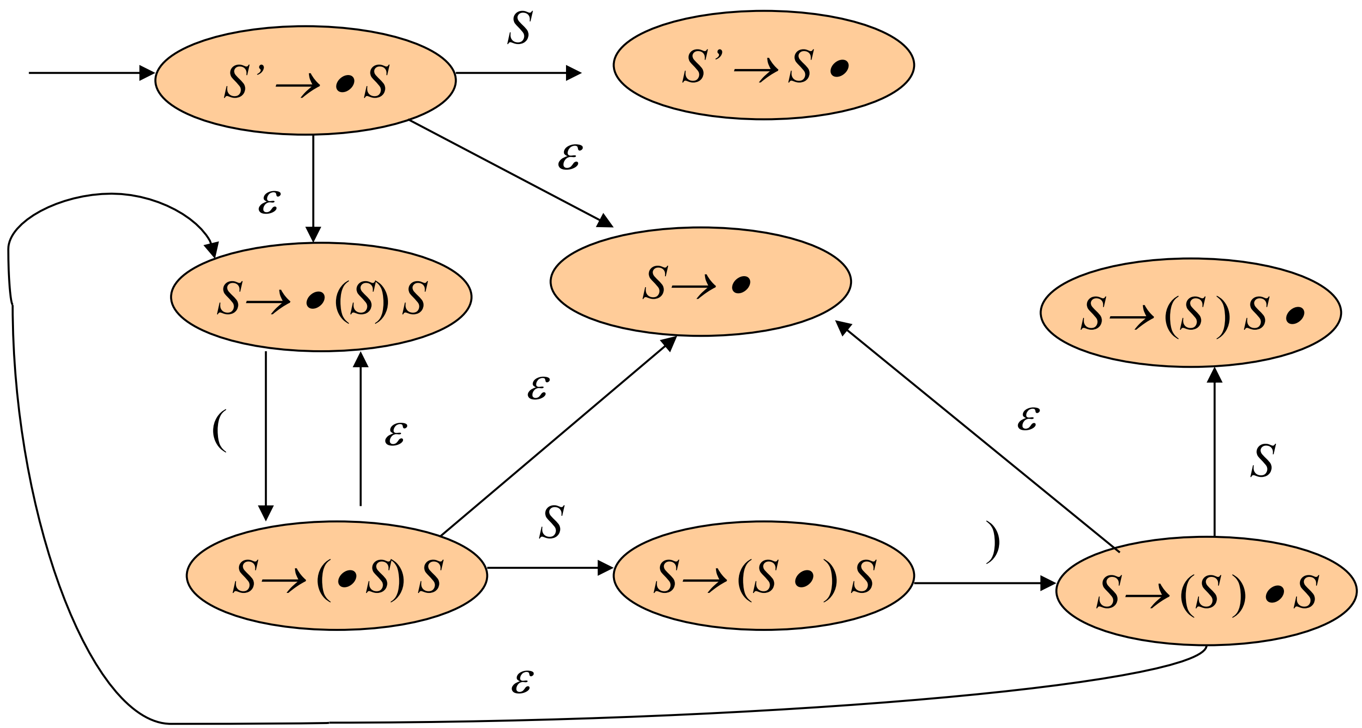
  any initial item $S \to \cdot\alpha$: a start state.
- The solution is to augment the grammar by a single production $S' \to S$,
- The initial item $S' \to \cdot S$: the *start state* of the NFA.
- The NFA will have some information on acceptance but not in the form of an accepting state.

5.2.2 Finite Automata of Items

- Example 5.5

  Note that every item in the figure with a dot before the nonterminal S has an ε-transition to every initial item of S.

$S' \rightarrow \cdot S$
$S' \rightarrow S\cdot$
$S \rightarrow \cdot(S)S$
$S \rightarrow (\cdot S)S$
$S \rightarrow (S\cdot)S$
$S \rightarrow (S)\cdot S$
$S \rightarrow (S)S\cdot$
$S \rightarrow \cdot$

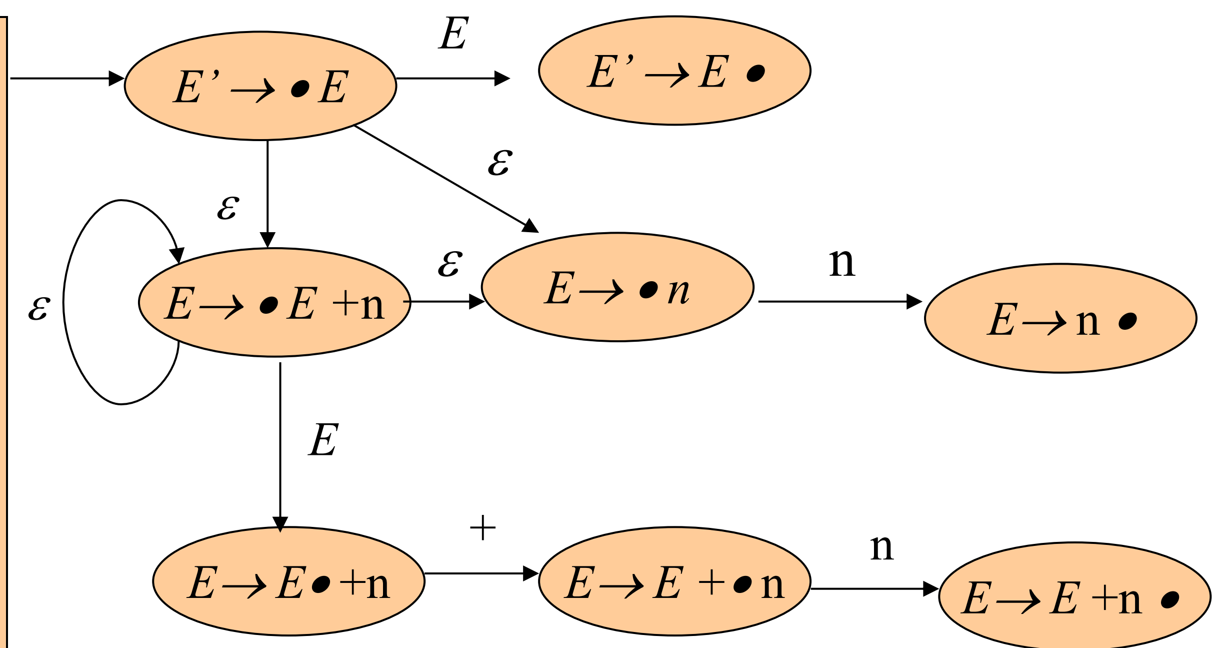


5.2.2 Finite Automata of Items

- Example 5.6

Note that the initial item $E \rightarrow .\ E + n$ has an ε-transition to itself (This situation will occur in all grammars with immediate left recursion.)

E’⟶·E
E’⟶E·
E⟶·E + n
E⟶E· + n
E⟶E +· n
E⟶E + n·
E⟶·n
E⟶n·



5.2.2 Finite Automata of Items

- Example 5.7 the DFA of the NFA of Figure 5.1.

$S' \rightarrow \cdot S$

$S' \rightarrow S\cdot$

$S \rightarrow \cdot(S)S$

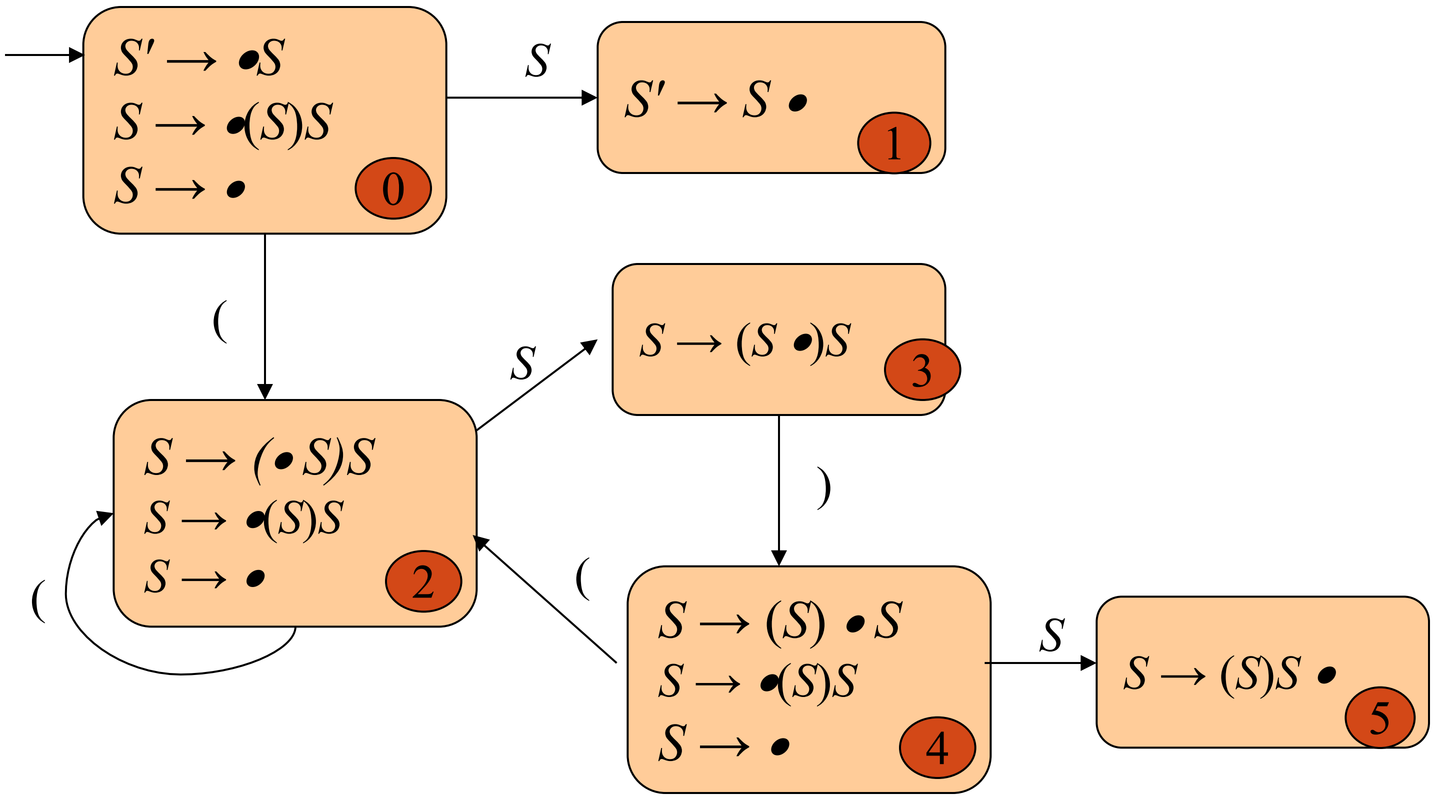
$S \rightarrow (\cdot S)S$

$S \rightarrow (S\cdot)S$

$S \rightarrow (S)\cdot S$

$S \rightarrow (S)S\cdot$

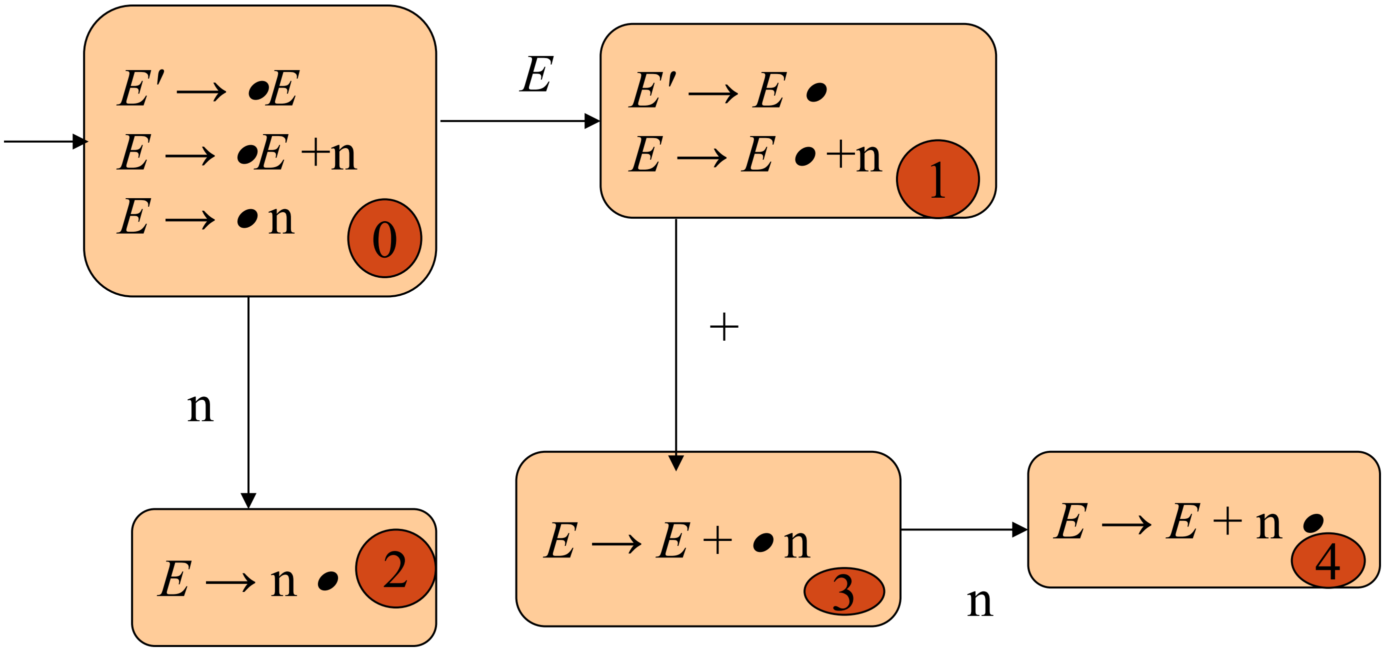
$S \rightarrow \cdot$



5.2.2 Finite Automata of Items

- Example 5.8 The DFA OF the NFA of Figure 5.2

$E' \rightarrow \cdot E$
$E' \rightarrow E \cdot$
$E \rightarrow \cdot E + n$
$E \rightarrow E \cdot + n$
$E \rightarrow E + \cdot n$
$E \rightarrow E + n \cdot$
$E \rightarrow \cdot n$
$E \rightarrow n \cdot$



5.2.3 The LR(0) Parsing Algorithm

- The parsing stack to store: *symbols* and *state numbers*.
- Pushing the new *state number* onto the parsing stack after each push of *a symbol*.

Parsing stack	Input
\$0	Input String
\$0 n2	Rest input string \$

Parsing stack	Symbol stack	Input
0	\$	Input String
02	\$n	Rest input string \$

5.2.3 The LR(0) Parsing Algorithm

- Definition

Let s be the current state (at the top of the parsing stack).Then actions are defined as follows:

1. If state s contains any item of the form $A \rightarrow \alpha{\cdot}X\beta$ (X is a terminal). Then the action is to *shift* the current input token on to the stack.

2. If state s contains any *complete item* (an item of the form $A \rightarrow \gamma\cdot$), then the action is to reduce by the rule $A \rightarrow \gamma\cdot$

- A *reduction* by the rule $S' \rightarrow S$, where S' is the start state,
- *Acceptance* if the input is empty
- *Error* if the input is not empty.

5.2.3 The LR(0) Parsing Algorithm

- A grammar is said to be LR(0) grammar if the above rules are unambiguous.
- A grammar is LR(0) ***if and only if***
  - Each state is a shift state(a state containing only “shift” items)
  - A reduce state containing a single complete item.

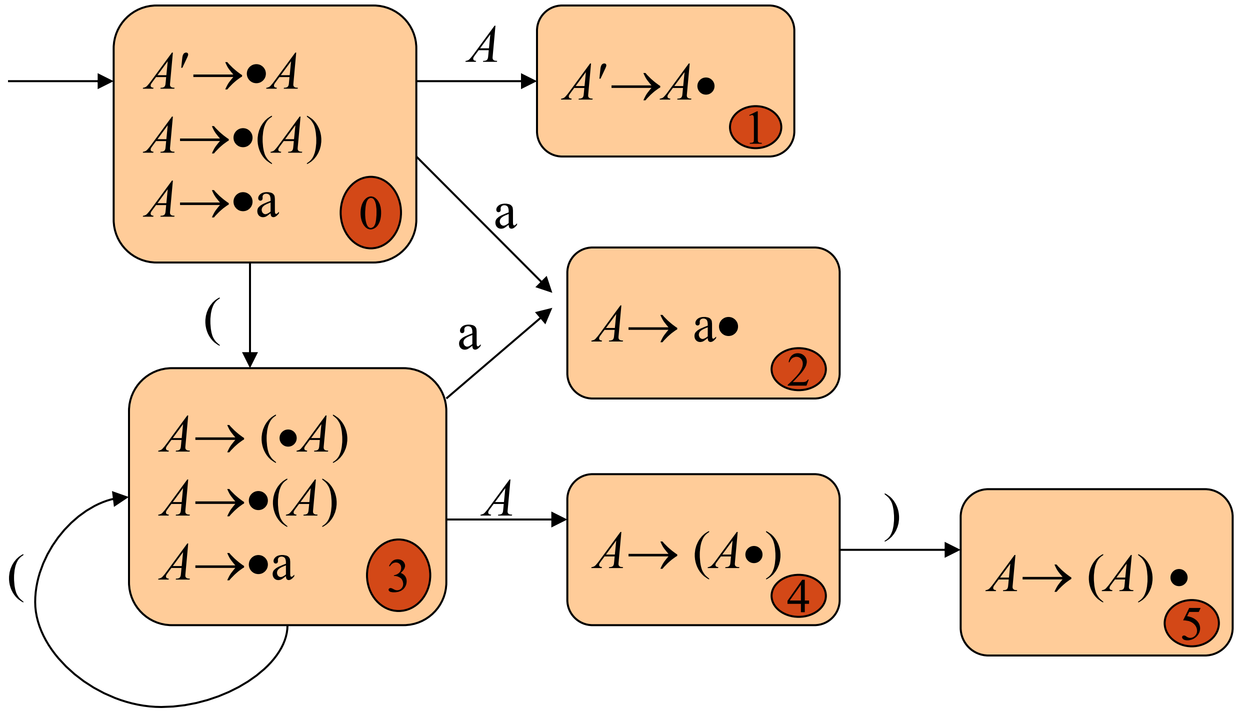
5.2.3The LR(0) Parsing Algorithm

Grammar:
$S \rightarrow (L) \mid id$
$L \rightarrow S \mid L, S$

Derivation	stack	input	action
((a),b) ⇐	1	((a),b)	shift, goto 3
((a),b) ⇐	13	(a),b)	shift, goto 3
((a),b) ⇐	133	a),b)	shift, goto 2
((a),b) ⇐	1332	),b)	reduce S→id
((S),b) ⇐	1337	),b)	reduce L→S
((L),b) ⇐	1335	),b)	shift, goto 6
((L),b) ⇐	13356	,b)	reduce S→(L)
(S,b) ⇐	137	,b)	reduce L→S
(L,b) ⇐	135	,b)	shift, goto 8
(L,b) ⇐	1358	b)	shift, goto 9
(L,b) ⇐	13582	)	reduce S→id
(L,S) ⇐	13589	)	reduce L→L , S
(L) ⇐	135	)	shift, goto 6
(L) ⇐	1356		reduce S→(L)
S	14		done

5.2.3 The LR(0) Parsing Algorithm

- Example 5.9 Consider the grammar $A \to (\,A\,) \mid a$



5.2.3The LR(0) Parsing Algorithm

	Parsing stack	Input	Action
1	\$0	((a))\$	Shift
2	\$0 (3	(a))\$	Shift
3	\$0 (3 (3	a))\$	Shift
4	\$0 (3 (3 a2	))\$	Reduce $A \rightarrow a$
5	\$0 (3 (3 A4	))\$	Shift
6	\$0 (3 (3 A4)5	)\$	Reduce $A \rightarrow (A)$
7	\$0 (3 A4	)\$	Shift
8	\$0 (3 A4)5	\$	Reduce $A \rightarrow (A)$
9	\$0 A1	\$	Accept

5.2.3 The LR(0) Parsing Algorithm

- The DFA of sets of items and the actions : be combined into a parsing table.
- The LR(0) parsing becomes *a table-driven parsing* method.
- The table rows labeled with the states of the DFA.
- The columns to be labeled with "*shift*" and "*reduce*".

State	Action	Rule	Input			Goto
			(	a	)	A
0	Shift		3	2		1
1	Reduce	$A' \rightarrow A$				
2	Reduce	$A \rightarrow (A)$				
3	Shift		3	2		4
4	Shift				5	
5	Reduce	$A \rightarrow a$				

5.2.3The LR(0) Parsing Algorithm

State	Input			Goto
	(	a	)	A
0	s3	s2		1
1	r1	r1	r1	
2	r2	r2	r2	
3	s3	s2		4
4			s5	
5	r3	r3	r3	