



Bottom

- Example 5.1

Consider the following augmented grammar for balanced Parentheses:

$$S' \rightarrow S$$

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

- Table 5.1 Parsing action of a bottom-up parser for the grammar of

	Parsing stack	Input	Action
1	\$	() \$	shift
2	\$ () \$	reduce $S \rightarrow \epsilon$
3	\$ (S) \$	shift
4	\$ (S)	\$	reduce $S \rightarrow \epsilon$
5	\$ (S) S	\$	reduce $S \rightarrow (S) S$
6	\$ S	\$	reduce $S' \rightarrow S$
7	\$ S'	\$	accept

root

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

Bottom-up

LR(0):R指從rule之right(右邊)看起

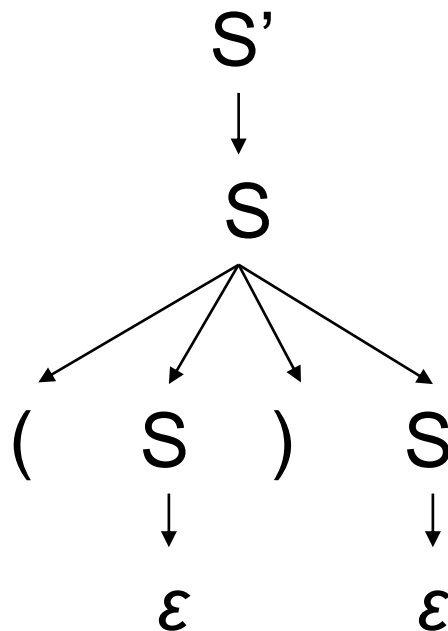
LR(1) add **Follow** set in **CFSM**

(powerful, but too many states)

To reduce the large number of states in LR(1):

(1) Simple LR(1) \rightarrow follow LR(0)+ Follow set

(2) LookAhead LR(1) \leftarrow follow LR(1)+merge the same core)



- Ex. 2

Consider the following augmented grammar for rudimentary arithmetic


expressions (no parentheses and one operation):

$$E' \rightarrow E$$

$$E \rightarrow E + n \mid n$$

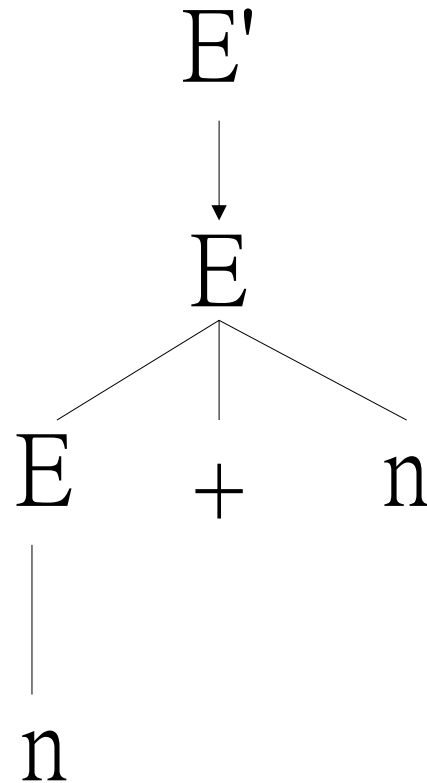
- A bottom-up parse of the string **$n + n$** using this is given in table 5.2

- Table 5.2 Parsing actions of a bottom-up parser for the grammar of

	Parsing stack	Input	Action	
1	\$	$n + n$ \$	shift	 root
2	\$ n	$+$ n \$	reduce $E \rightarrow n$	
3	\$ E	$+$ n \$	shift	
4	\$ $E +$	n \$	shift	
5	\$ $E + n$	\$	reduce $E \rightarrow E + n$	
6	\$ E	\$	reduce $E' \rightarrow E$	
7	\$ E'	\$	accept	

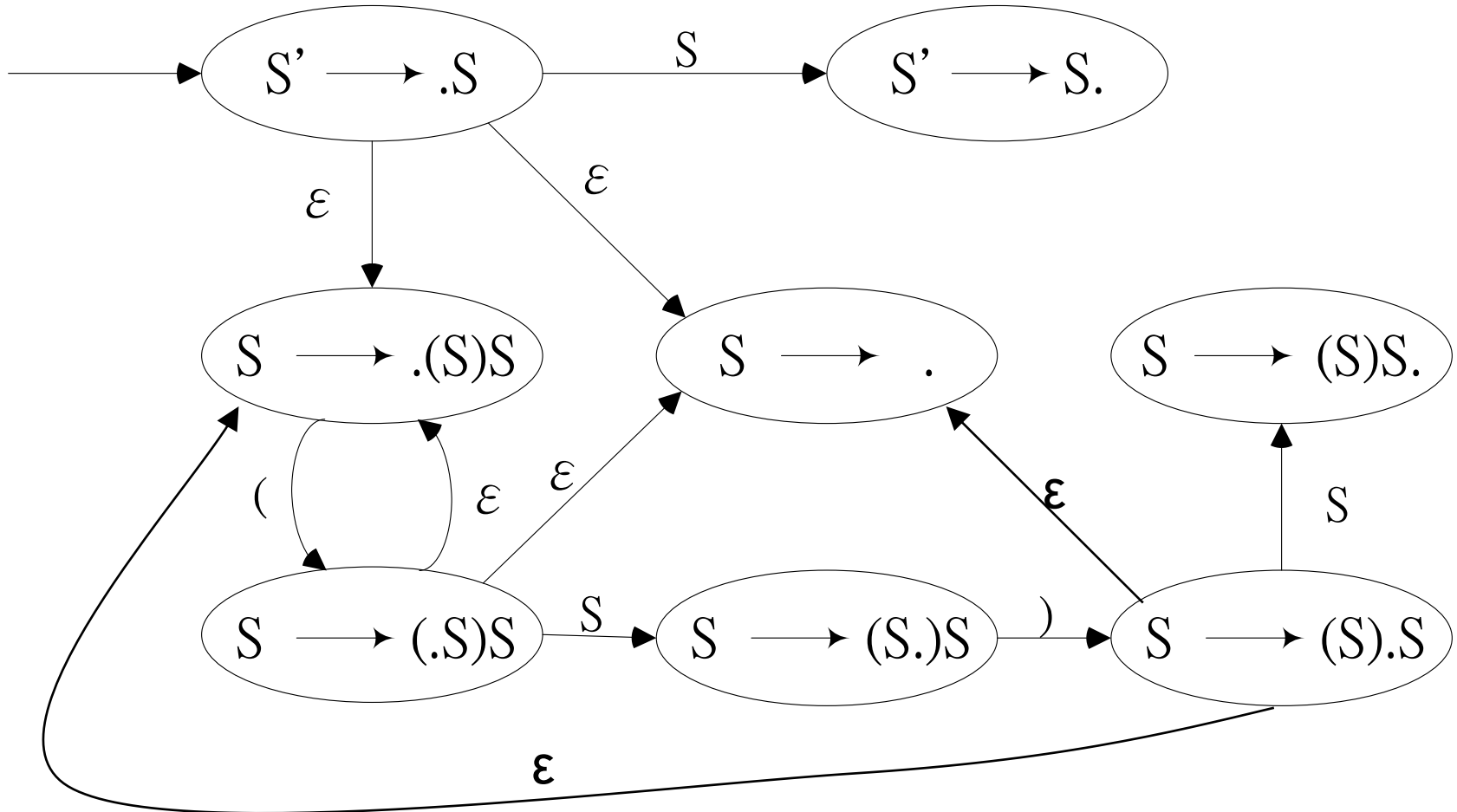
example 5.2

- $E' \Rightarrow E \Rightarrow E + n \Rightarrow n + n$



$$\begin{aligned}
 S' &\longrightarrow S \\
 S &\longrightarrow (S)S \mid \varepsilon
 \end{aligned}$$

$$\begin{aligned}
 S' &\longrightarrow \cdot S \\
 S' &\longrightarrow S \cdot \\
 S &\longrightarrow \cdot (S)S \\
 S &\longrightarrow (\cdot S)S \\
 S &\longrightarrow (S \cdot)S \\
 S &\longrightarrow (S) \cdot S \\
 S &\longrightarrow (S)S \cdot \\
 S &\longrightarrow \cdot
 \end{aligned}$$



8 states
Ex. 5.1

$$E' \rightarrow E$$

$$E \rightarrow E+n \mid n$$

$$E' \longrightarrow .E$$

$$E' \longrightarrow E.$$

$$E \longrightarrow .E + n$$

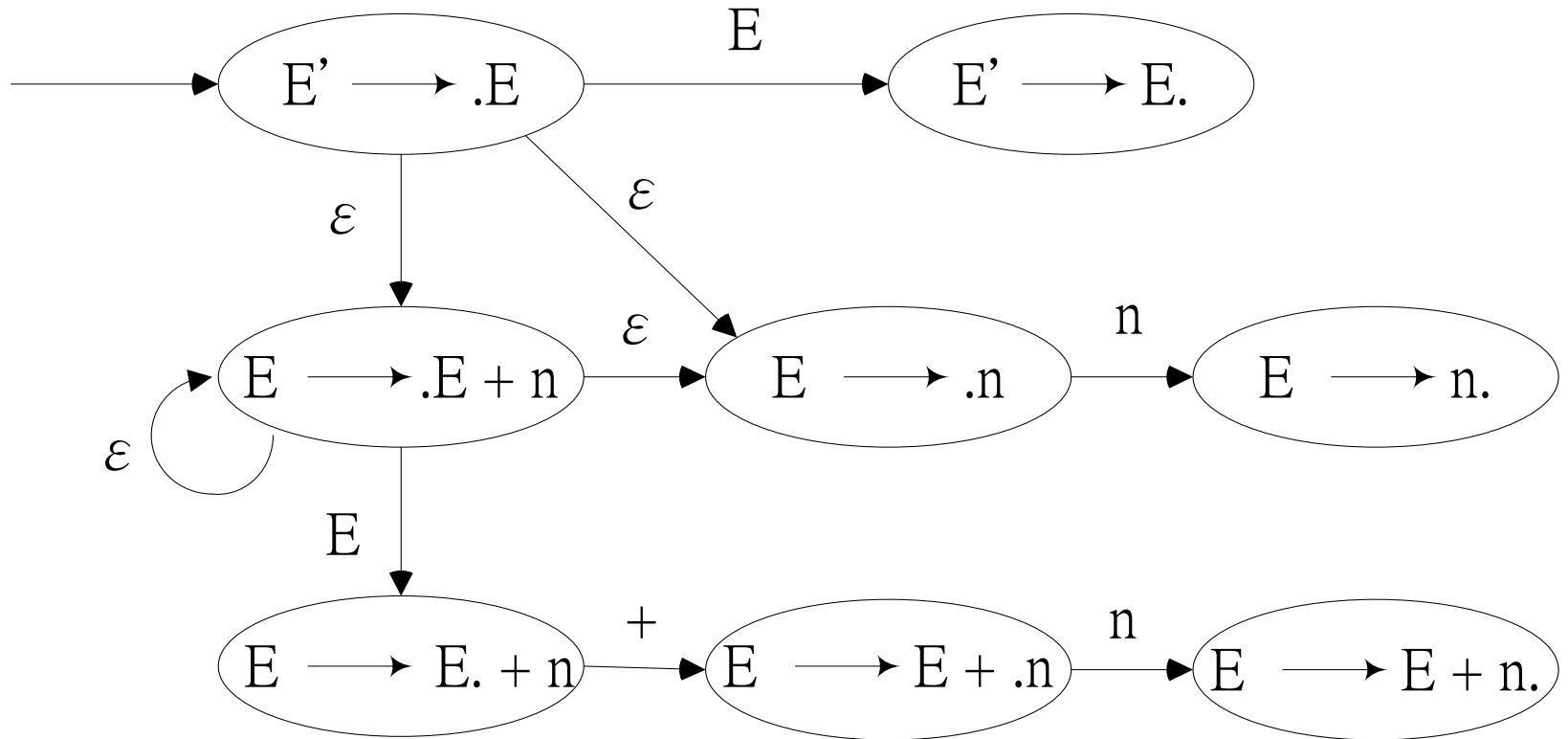
$$E \longrightarrow E. + n$$

$$E \longrightarrow E + .n$$

$$E \longrightarrow E + n.$$

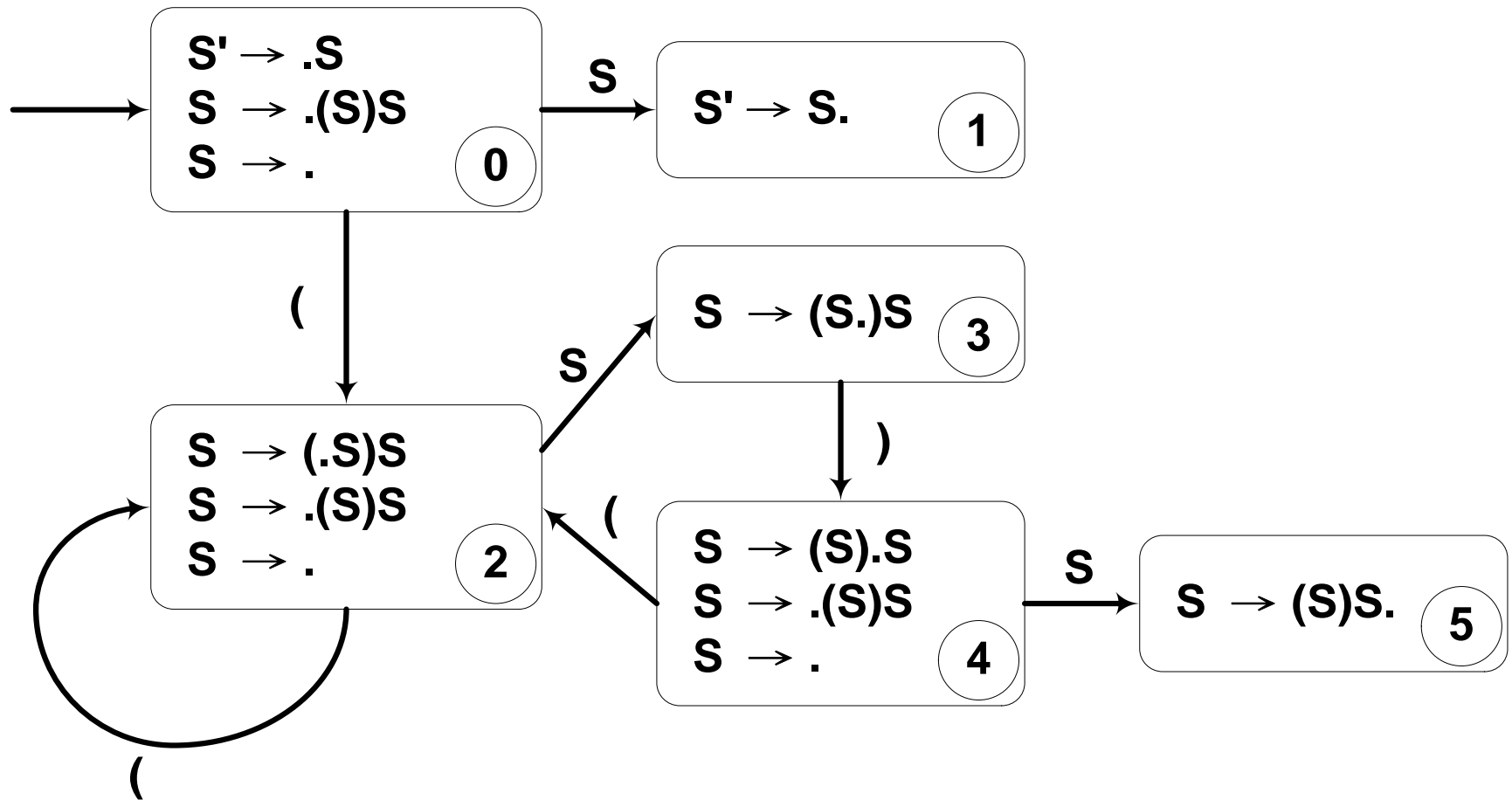
$$E \longrightarrow .n$$

$$E \longrightarrow n.$$



8 states
Ex. 5.2

- 一個closure為一個state.



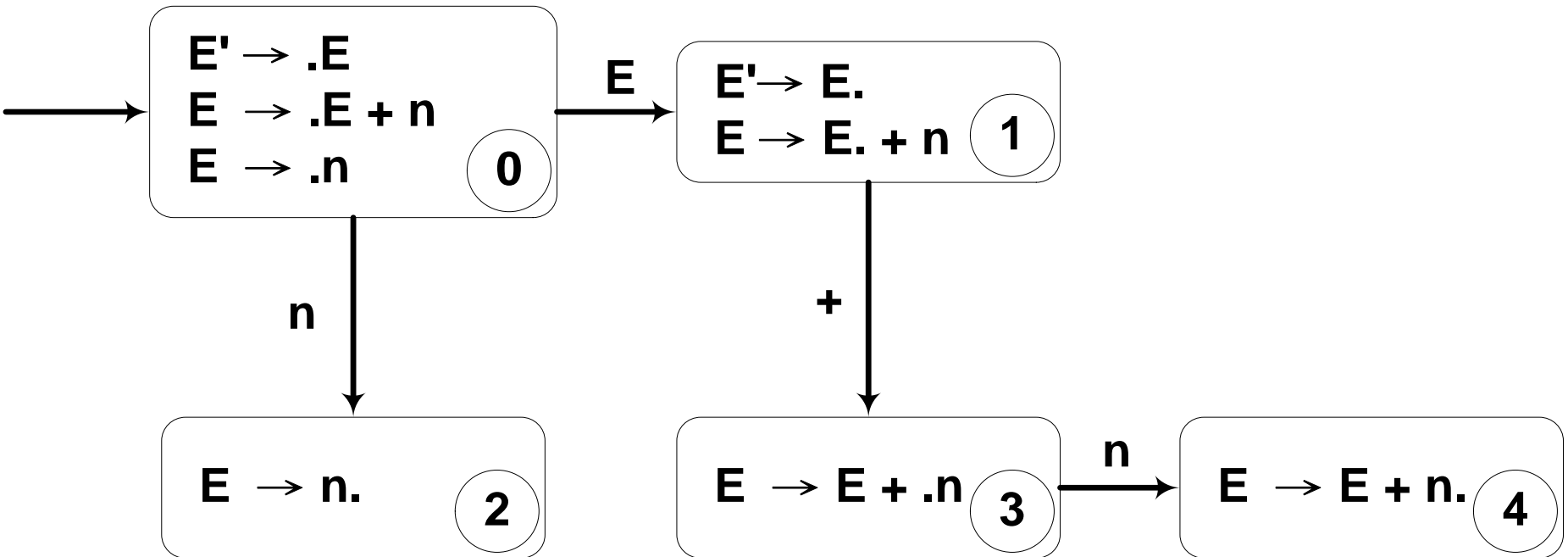
■ GoTo

	0	1	2	3	4	5
(2		2		2	
)				4		
S	1		3		5	

■ Action

State	0	1	2	3	4	5
action	S R3	R1	S R3	S	S R3	R2

Shift/Reduce conflict, 非 LR(0)



■ GoTo

	0	1	2	3	4
n	2			4	
+		3			
E	1				

■ Action

State	0	1	2	3	4
action	S	S R1	R3	S	R2

Shift-Reduce conflict, 非 LR(0)

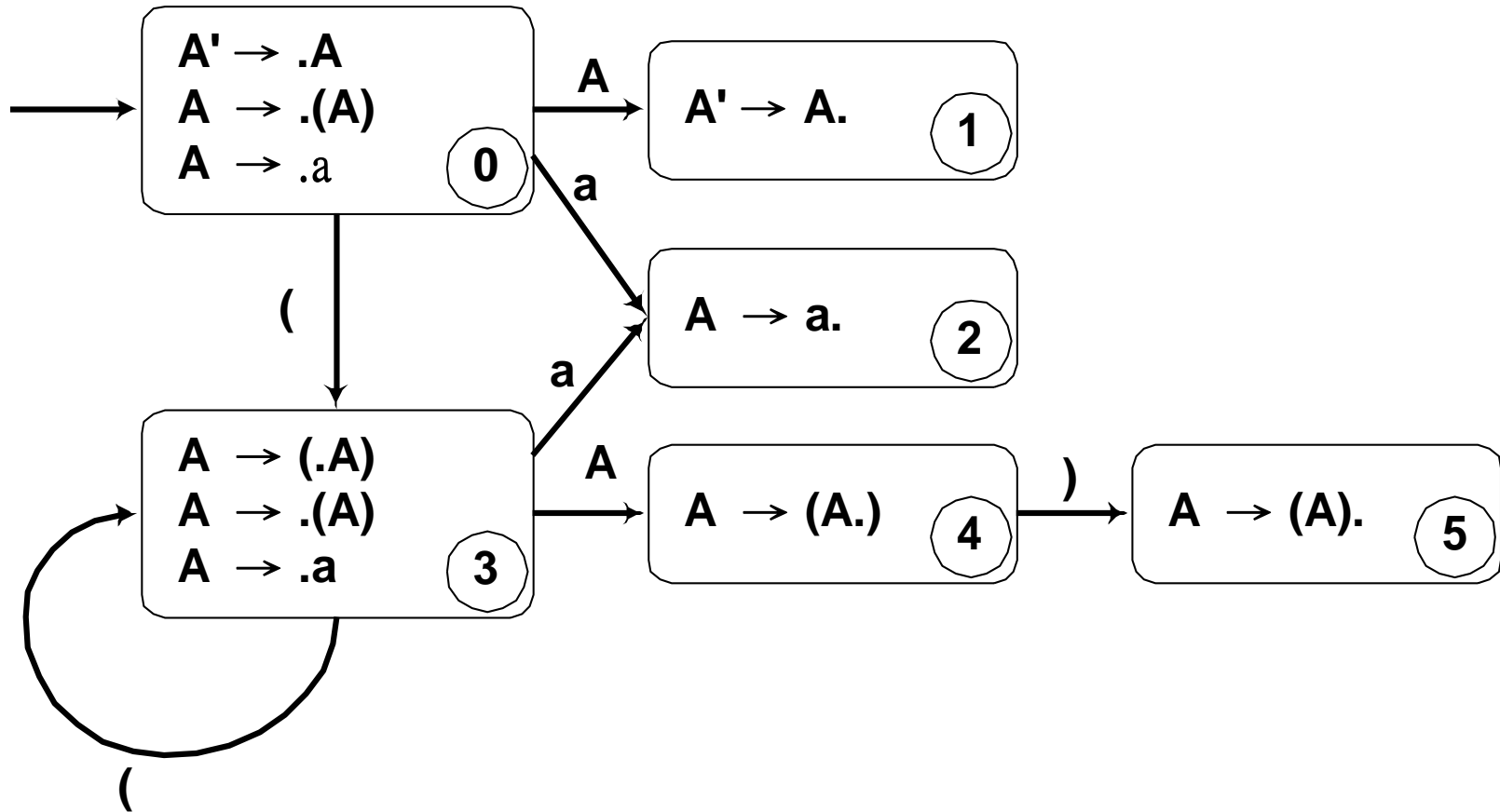
- *The LR(0) parsing algorithm.* Let s be the current state (at the top of the parsing stack). Then actions are defined as follows:
 1. If state s contains any item of the form $A \rightarrow \alpha.X\beta$, where X is a terminal, then the action is to shift the current input token onto the stack. If this token is X , and state s contains item $A \rightarrow \alpha.X\beta$, then the new state to be pushed on the stack is the state containing the item $A \rightarrow \alpha X.\beta$. If this token is not X for some item in state s of the form just described, an error is declared.

2. If state s contains any complete item (an item of form $A \rightarrow \alpha_1.$), then the action is to reduce by the rule $A \rightarrow \alpha_1$. A reduction by the rule $S' \rightarrow S$, where S' is the start state, is equivalent to acceptance, provided the input is empty, and error if the input is not empty. In all other cases, the new state is computed as follows. Remove the string α_1 and all of its corresponding states from the parsing stack (the string α must be at the top of the stack, according to the way the DFA is constructed). Correspondingly, back up in the DFA to the state from which the construction of α began (this must be the state uncovered by the removal of α_1). Again, by the construction of the DFA, this state must contain an item of the form $B \rightarrow \alpha_2.A\beta$. Push A onto the stack, and push (as the new state) the state containing the item $B \rightarrow \alpha_2.A\beta$. (Note that this corresponds to following the transition on A in the DFA, which is indeed reasonable, since we are pushing A onto the stack.)

- A grammar is said to be an **LR(0) grammar** if the above rules unambiguous. This means that if a state contains a complete item $A \rightarrow \alpha.$, then it can contain no other items. Indeed, if such a state also contains a “shift” item $A \rightarrow \alpha.X\beta.$ (X is a terminal), then an ambiguity arises as to whether action (1) or action (2) is to be performed. This situation is call a **shift-reduce conflict**. Similarly, if such a state contains another complete item $B \rightarrow \beta.$, then an ambiguity arises as to which production to use for the reduction ($A \rightarrow \alpha.$ or $B \rightarrow \beta.$). This situation is called a **reduce-reduce conflict**. Thus, a grammar is LR(0) if and only if each state if a shift state (a state containing only “shift” items) or a reduce state containing a single complete item.

■ $A' \rightarrow A$

$A \rightarrow (A)a$



■ GoTo

	0	1	2	3	4	5
(3			3		
a	2			2		
A	1			4		
)					5	

■ Action

State	0	1	2	3	4	5
action	S	A	R3	S	S	R2

LR(0)

■ Finite Automata of LR(0) items and LR(0) Parsing

$A' \rightarrow A$
 $A \rightarrow (A)$
 $A \rightarrow a$

	Parsing stack	Input	Action
1	\$0	((a)) \$	shift
2	\$0 (3	(a)) \$	shift
3	\$0 (3 (3	a)) \$	shift
4	\$0 (3 (3 a 2)) \$	reduce $A \rightarrow a$; goto(3,A)=4
5	\$0 (3 (3 A 4)) \$	shift
6	\$0 (3 (3 A 4)5) \$	reduce $A \rightarrow (A)$; goto(3,A)=4
7	\$0 (3 A 4) \$	shift
8	\$0 (3 A 4)5	\$	reduce $A \rightarrow (A)$; goto(0,A)=1
9	\$0 A1	\$	Accept

State	Action	Rule	Input			Goto
			(a)	A
0	shift		3	2		1
1	accept	$A' \rightarrow A$				
2	reduce	$A \rightarrow a$				
3	shift		3	2		4
4	shift				5	
5	reduce	$A \rightarrow (A)$				