

Grammar Class Inclusion Tree

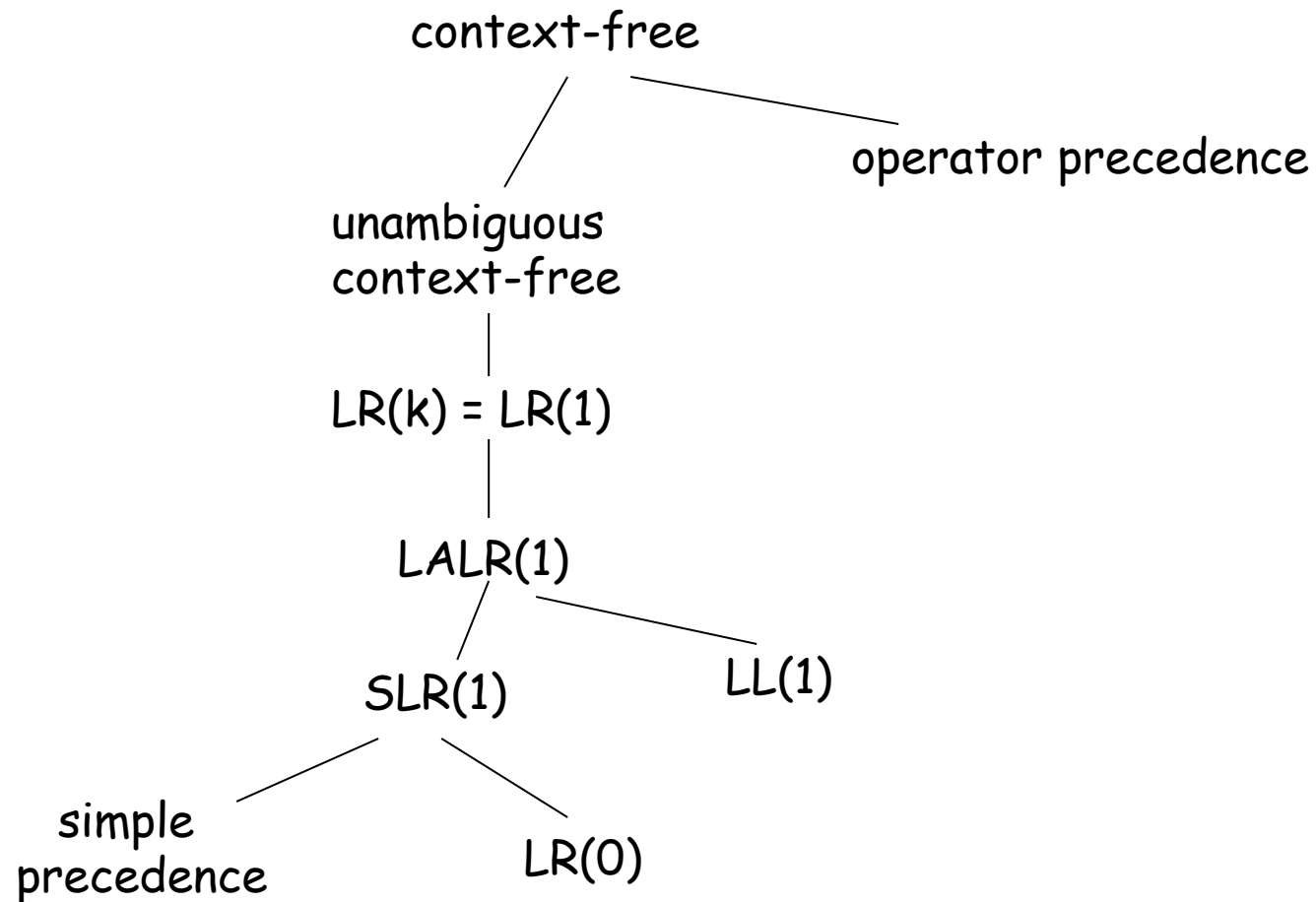


Table-driven Bottom-up Parsing

- Start at the leaves and grow toward root
- Bottom-up parsers handle a large class of grammars
- Most prevalent is based on LR(k)
- Why LR Parsing ?
 - Recognize many programming languages
 - Detect Syntax Errors
 - No backtracking

Bottom Up Parsing

Shift and Reduce

Consider: $S \rightarrow bMb$
 $M \rightarrow (L \mid a$
 $L \rightarrow M a)$

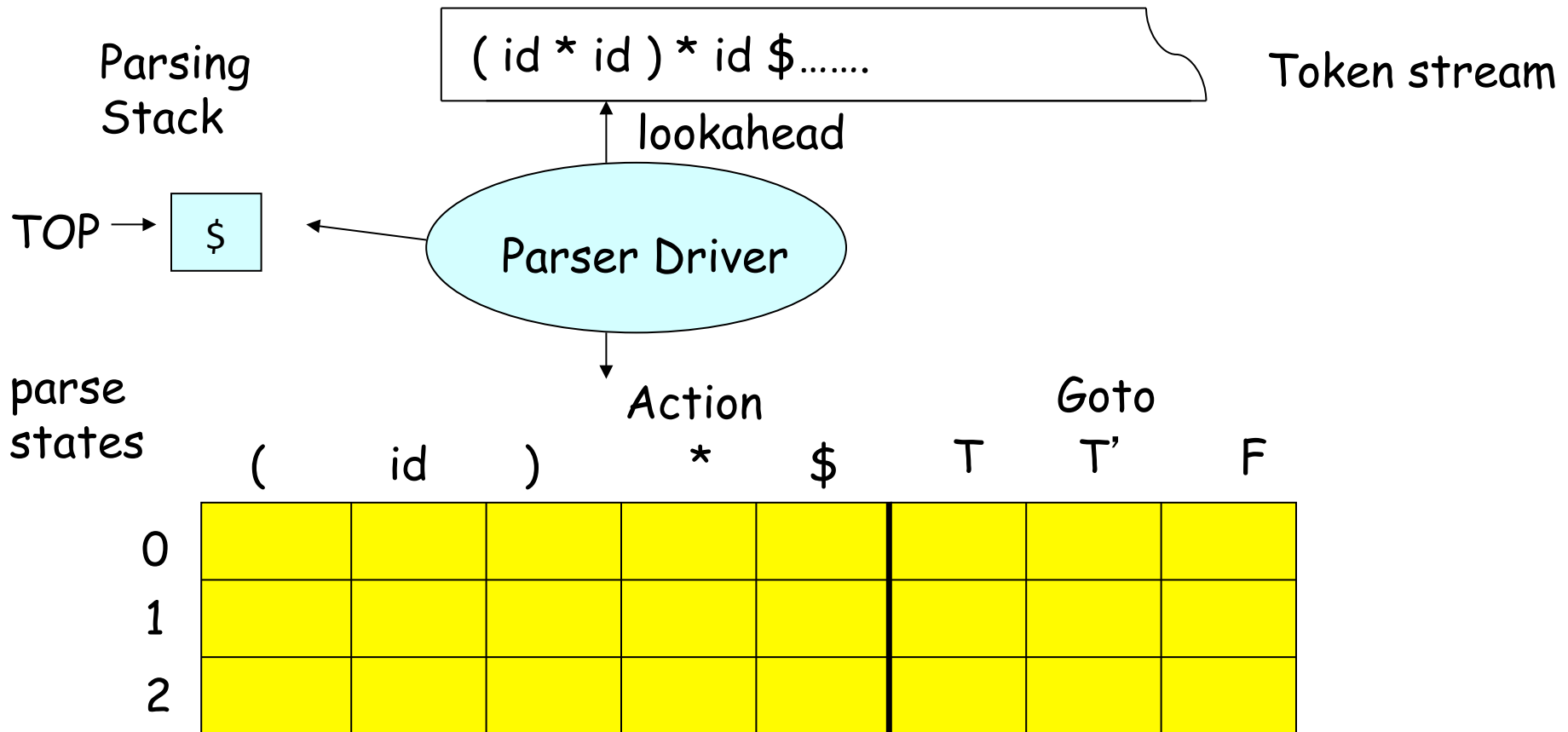
String: bab

<u>Stack:</u>	<u>Input and Current Lookahead:</u>
\$	bab\$

Handle: a substring β of the tree's frontier that

- matches some production $A \rightarrow \beta$ that occurs as one step in the rightmost derivation

Table-driven Bottom-up Parsing Overview



Table[state,terminal] =

- shift token and state onto stack.
- reduce by production $A \rightarrow \beta$
 pop rhs from stack; push A ; push next state
 given by $Goto[\text{exposed state}, A]$
- accept
- error

LR Parsing Example

- 1: $P \rightarrow b S e$
- 2: $S \rightarrow a ; S$
- 3: $S \rightarrow b S e ; S$
- 4: $S \rightarrow \epsilon$

Parse Table

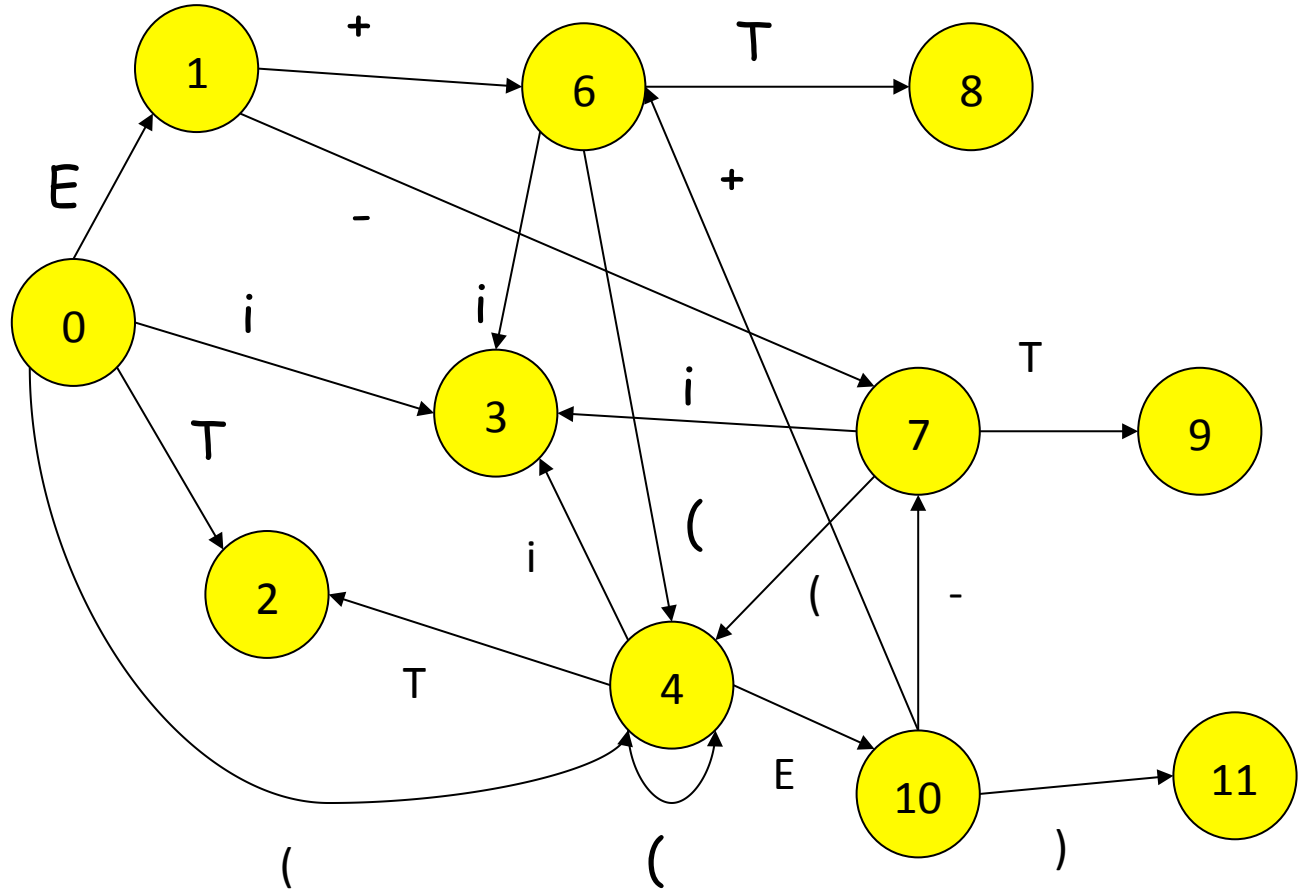
Stack	Input	state	b	e	a	;	\$	P	S
		0		s1					
0	ba;a;e\$	1		s4	r4	s5			2
0b1	a;a;e\$	2			s3				
0b1a5	;a;e\$	3					accept		
0b1a5;6	a;e\$	4		s4	r4	s5			7
0b1a5;6a5	;e\$	5					s6		
0b1a5;6a5;6	e\$	6		s4	r4	s5			10
0b1a5;6a5;6S10	e\$	7			s8				
0b1a5;6S10	e\$	8					s9		
0b1S2	e\$	9		s4	r4	s5			11
0b1S2e3	\$	10			r2				
accept!		11			r3				

DFA for Parser

$S \rightarrow E$
 $E \rightarrow T \mid E + T \mid E - T$
 $T \rightarrow I \mid (E)$

Reduce States:

3: $T \rightarrow i$
 2: $E \rightarrow T$
 8: $E \rightarrow E + T$
 9: $E \rightarrow E - T$
 11: $T \rightarrow (E)$
 1: (on \$) $S \rightarrow E$



<u>stack</u>	<u>input</u>
0	$i-(i+i)\$$
0i3	$-(i+i)\$$
0T2	$-(i+i)\$$
...	

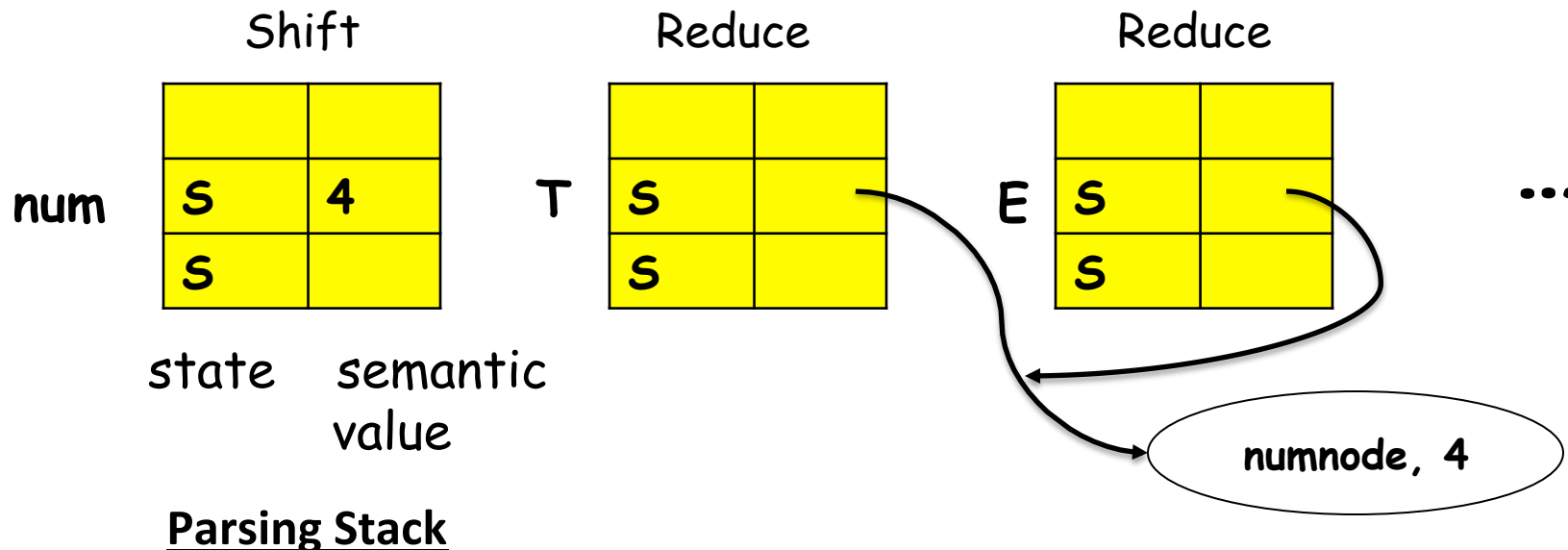
Semantic Actions during Parsing

BISON (Parser generator) example definition:

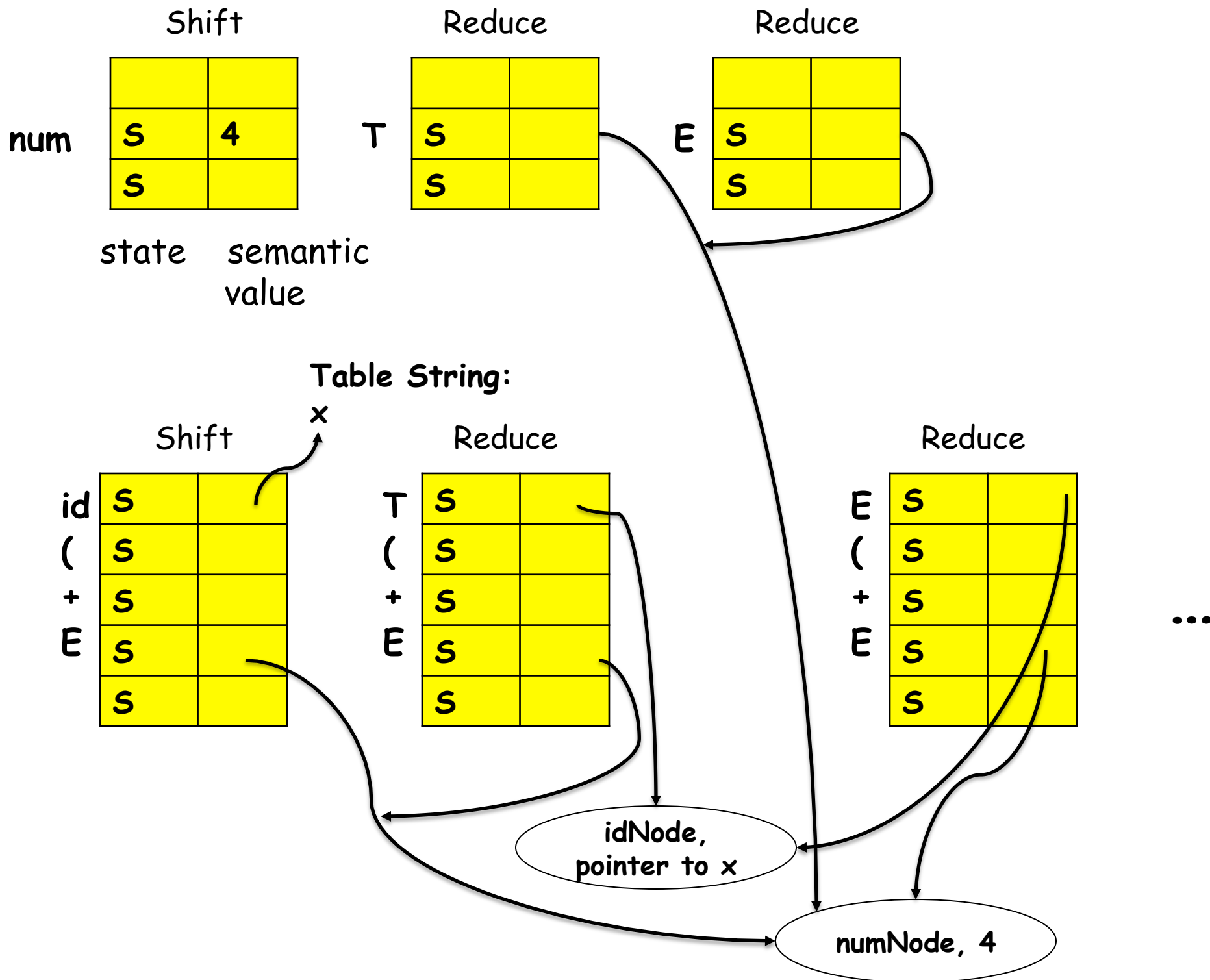
```

S -> E      { $$ = $1; root = $$; }
E -> E + T  { $$ = makenode('+', $1, $3); } // E is $1, + is $2, T is $3
E -> E - T  { $$ = makenode('-', $1, $3); }
E -> T      { $$ = $1; } // $$ is top of stack
T -> ( E )  { $$ = $2; }
T -> id     { $$ = makeleaf('idnode', $1); }
T -> num    { $$ = makeleaf('numnode', $1); }
    
```

Now Consider parsing: **4 + (x - y)**



Consider parsing: $4 + (x - y)$



Building LR(0) and SLR(1) Parse Tables

1. Augment grammar

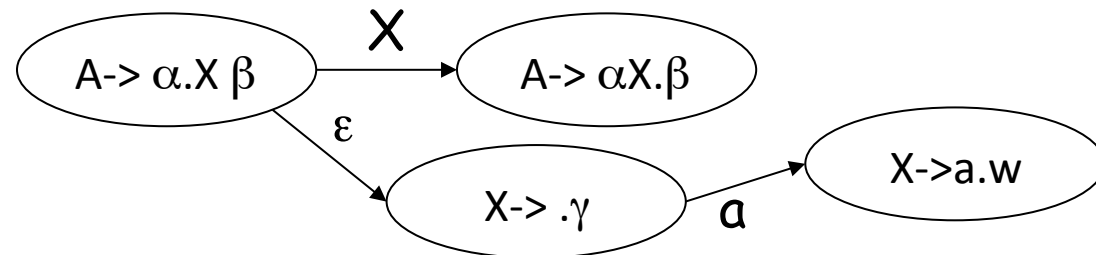
- Add a production $S' \rightarrow S$, where S is original start state
- Causes one ACCEPT table entry when reduce $S' \rightarrow S$ on \$.

2. Create DFA from grammar

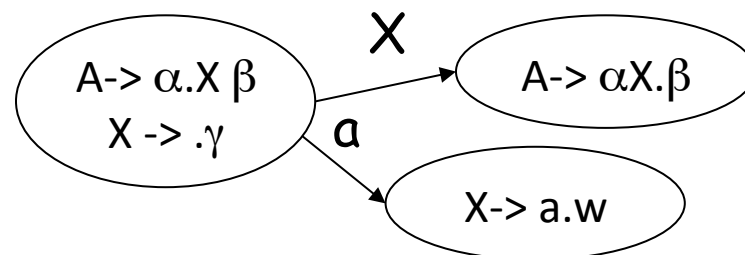
item $A \rightarrow \alpha . \beta$

- just seen a string derivable from α
- expect to see a string derivable from β

NFA: Each state represents a set of recognized viable prefixes (kernel set of items)



DFA: Subset construction to go from NFA to DFA = closure(kernel)



Items and States

LR(0) item - of a grammar G is a production of G with a dot at some position of the body

For example: $A \rightarrow XbZ$

All possible items are:

$A \rightarrow \cdot XbZ$

$A \rightarrow X \cdot bZ$

$A \rightarrow Xb \cdot Z$

$A \rightarrow XbZ \cdot$

Closure(item set I)

Given a set of kernel items I for a DFA state,

$$\text{Closure}(I) = \left\{ \begin{array}{l} \text{kernel items } I \\ \text{if } A \rightarrow \alpha.B\beta \text{ in } I \text{ and } B \rightarrow \gamma \\ \text{then add } B \rightarrow .\gamma \text{ to } I \end{array} \right.$$

Intuitively, we expect to see strings derivable from all nonterminals immediately to the right of the dot in any item in I.

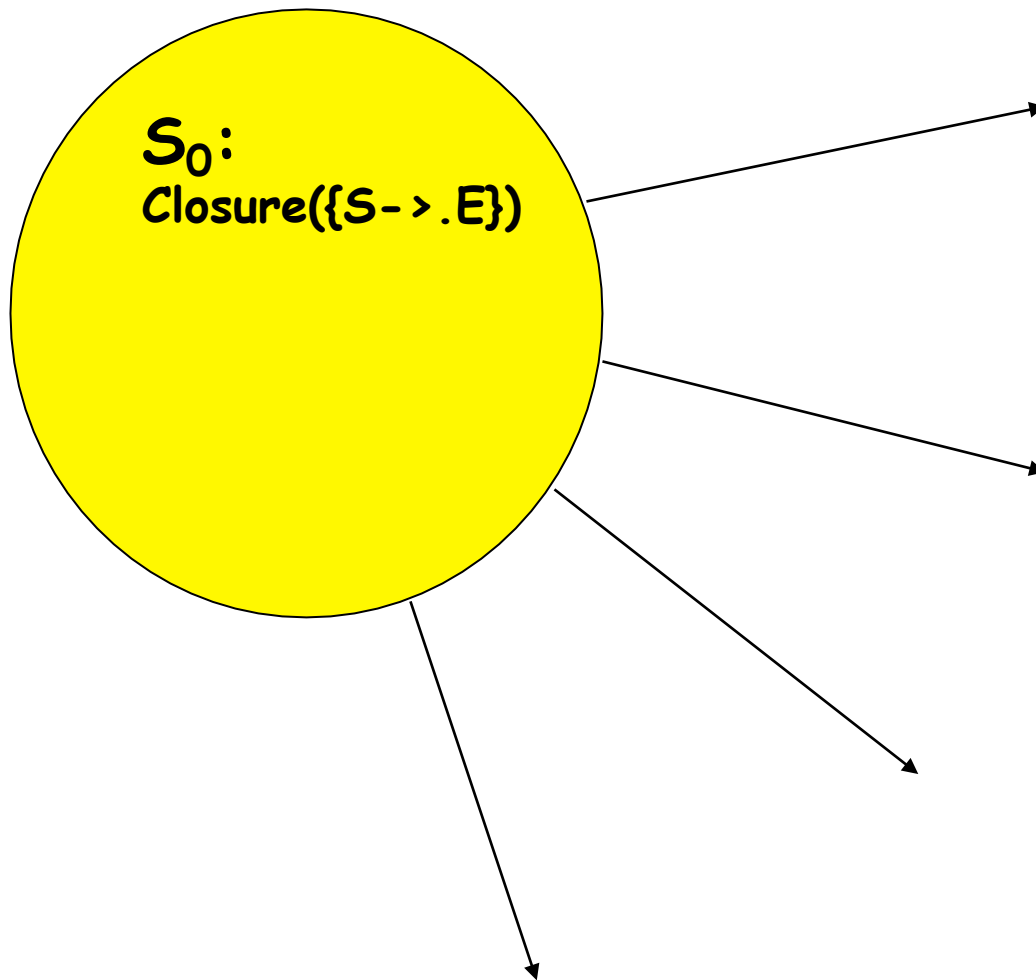
Example: $S \rightarrow E$
 $E \rightarrow T \mid E + T \mid E - T$
 $T \rightarrow i \mid (E)$

Let $I = \{S \rightarrow .E\}$
Closure(I) =

Let $I = \{E \rightarrow E+.T\}$
Closure(I) =

Example of DFA Construction

Grammar: $S \rightarrow E$
 $E \rightarrow T \mid E + T \mid E - T$
 $T \rightarrow i \mid (E)$



DFA Construction Algorithm

$S_0 = \text{Closure}(\{S' \rightarrow .S\});$

$\text{Todo} = \{S_0\};$

WHILE Todo not empty DO

 Remove an item set (ie, state) S_i from Todo ;

 FOR each grammar symbol X DO

 FOR each $A \rightarrow \alpha.X\beta$ in S_i DO

$S_{\text{new}} = \text{Closure}(A \rightarrow \alpha X .\beta);$

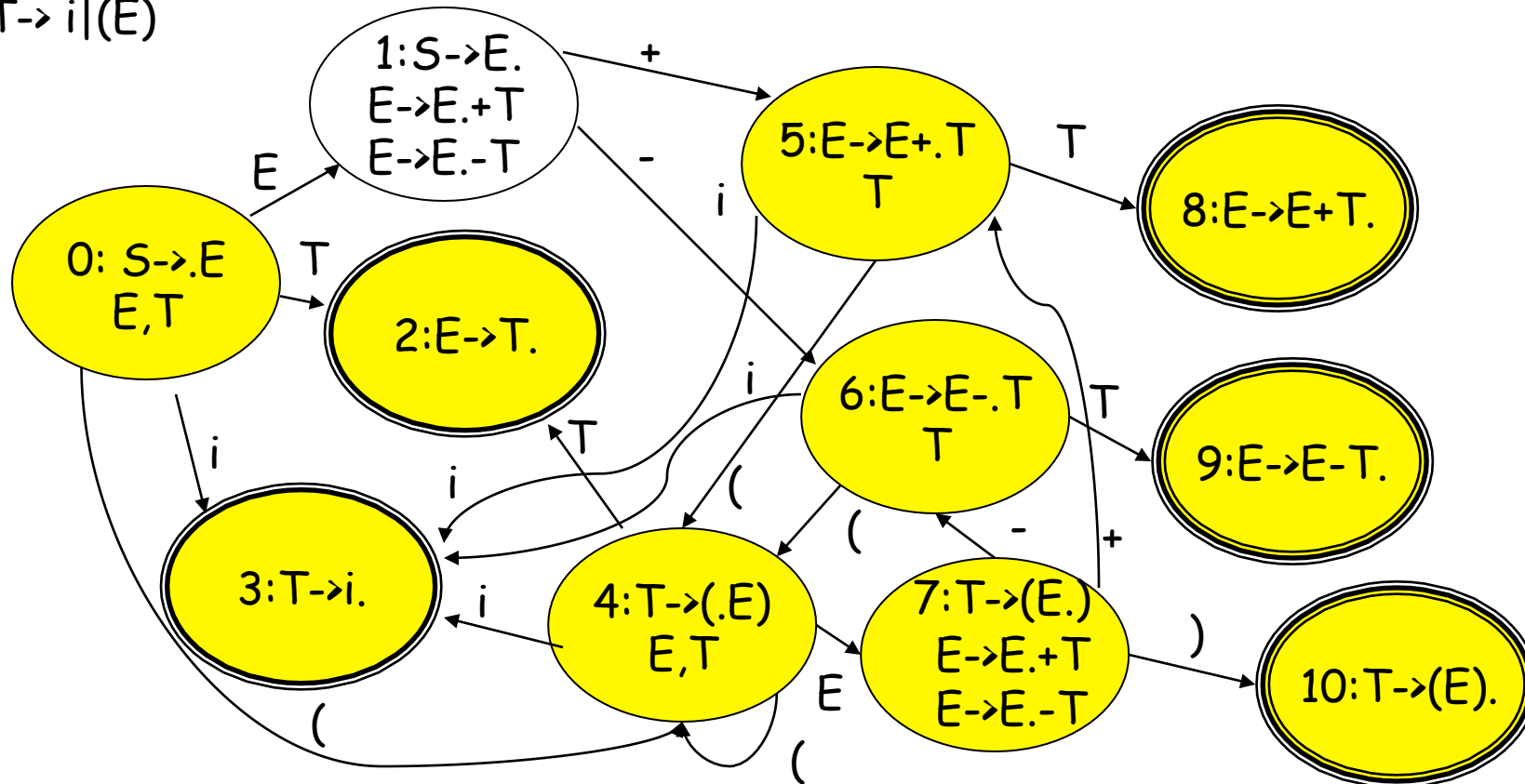
 If S_{new} is unique thus far,

 then Add S_{new} to DFA Add S_{new} to
 Todo ;

 Add edge $S_i \rightarrow S_{\text{new}}$ labeled by X

Final DFA for Example

$S \rightarrow E$
 $E \rightarrow T \mid E+T \mid E-T$
 $T \rightarrow i \mid (E)$

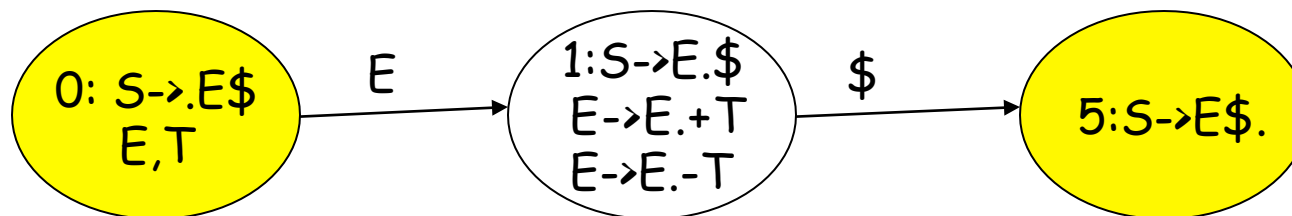


LR(0) grammar = DFA with no inadequate states, where inadequate state has shift/reduce or reduce/reduce conflict (e.g., state 1 is inadequate above)

SLR(1) grammar = Can resolve any inadequate states by FOLLOW info:
 $A \rightarrow \alpha \cdot$ and $B \rightarrow \beta \cdot X \delta$ in same state, but $FOLLOW(A) \cap \{X\}$ is empty.
 $A \rightarrow \alpha \cdot$ and $B \rightarrow \beta \cdot$ in same state, but $FOLLOW(A) \cap FOLLOW(B) = \emptyset$

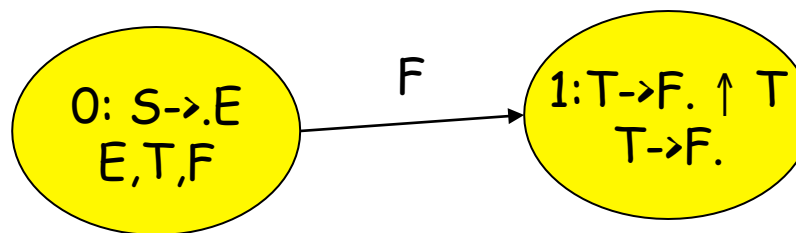
Examples showing LR(0) versus SLR(1)

To convert previous grammar to LR(0): Replace $S \rightarrow E$ by $S \rightarrow E\$$



Now consider Grammar:

$S \rightarrow E$
 $E \rightarrow E - T \mid T$
 $T \rightarrow F \uparrow T \mid F$
 $F \rightarrow (E) \mid i$



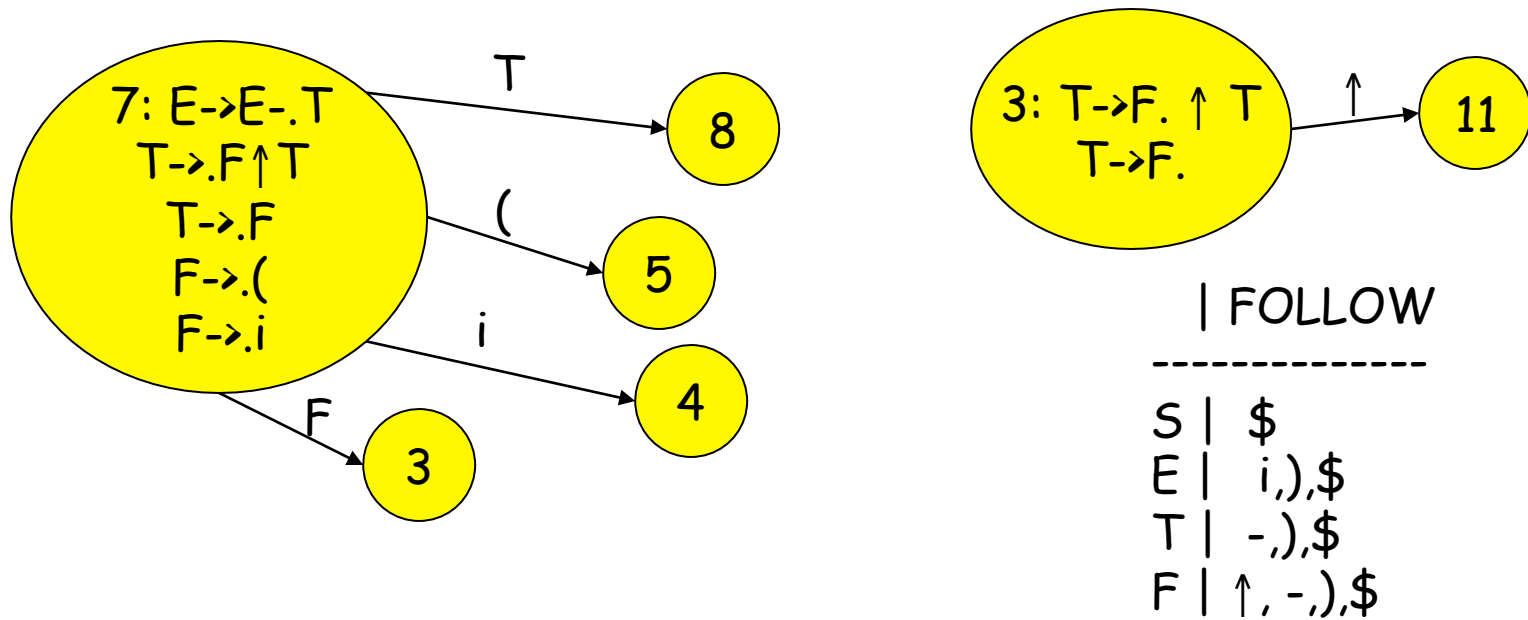
State 1 is inadequate, so **not LR(0)**

Is it SLR(1)?

- $FOLLOW(T) = \{-, \cdot, \$\}$

- $FOLLOW(T) \cap \{\uparrow\}$ is empty, so it is **SLR(1) Grammar**

From DFA to SLR(1) Parse Table



	FOLLOW
S	\$
E	i), \$
T	-,), \$
F	↑, -,), \$

state	i	-	↑	()	\$		S	E	T	F
3											
7											

Is the grammar LR(0), SLR(1)?

LR(0):

- construct parse table with no lookahead/FOLLOW info
If there are no multidefined entries, then LR(0)
- construct DFA. If there are no inadequate states, then LR(0).

SLR(1):

- construct parse table with FOLLOW info
If there are no multidefined entries, then SLR(1)
- construct DFA. If there are no inadequate states, or
for each inadequate state of the form:

$A \rightarrow \alpha.$
 $B \rightarrow \beta.$

$\text{FOLLOW}(A) \cap \text{FOLLOW}(B)$ is empty,

AND

$A \rightarrow \alpha.$
 $B \rightarrow \beta.a\gamma$

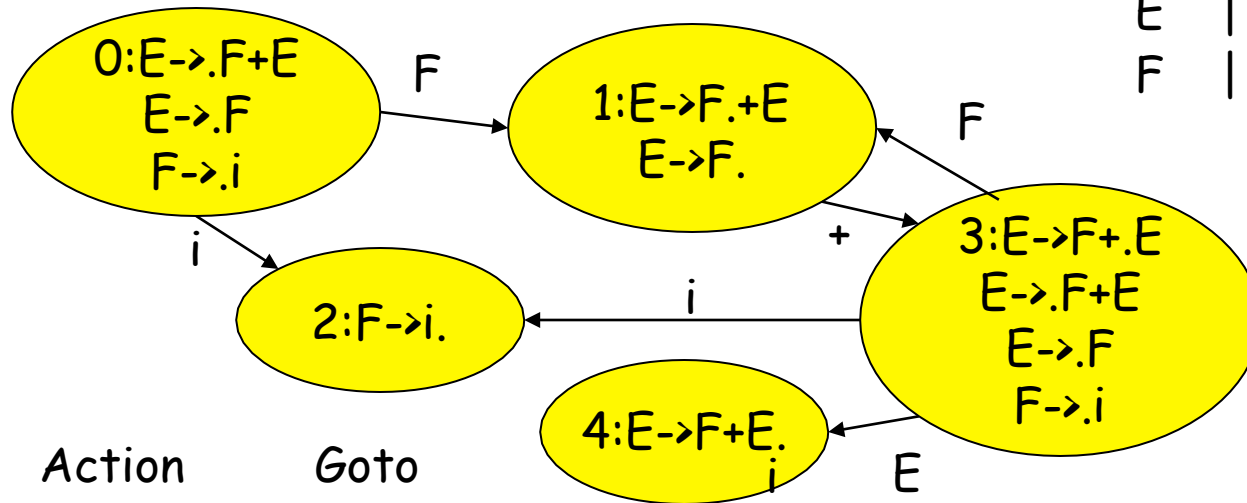
$\text{FOLLOW}(A) \cap \{a\}$ is empty

THEN SLR(1)

Why augment the grammar?

Consider $E \rightarrow F + E \mid F$
 $F \rightarrow i$

FOLLOW		
E		\$
F		\$, +



	Action			Goto	
	+	i	\$		E F
0	s2				1
1	s3	?			
2	r3		r3		
3	s2				4 1
4		?			

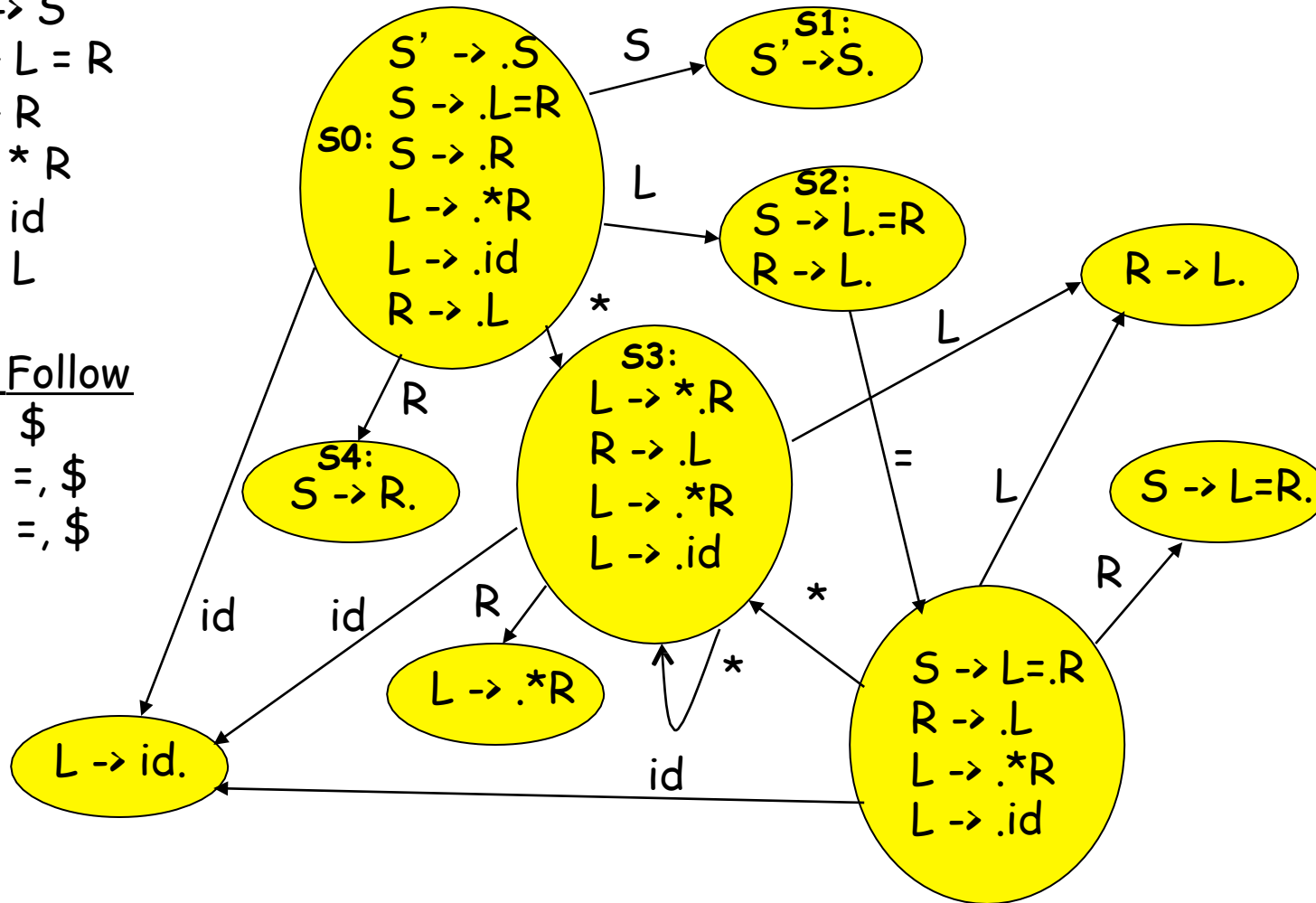
? Reduce $E \rightarrow F$ or Accept

? Reduce $E \rightarrow F + E$ or Accept

Example - not SLR(1)

$S' \rightarrow S$
 $S \rightarrow L = R$
 $S \rightarrow R$
 $L \rightarrow * R$
 $L \rightarrow id$
 $R \rightarrow L$

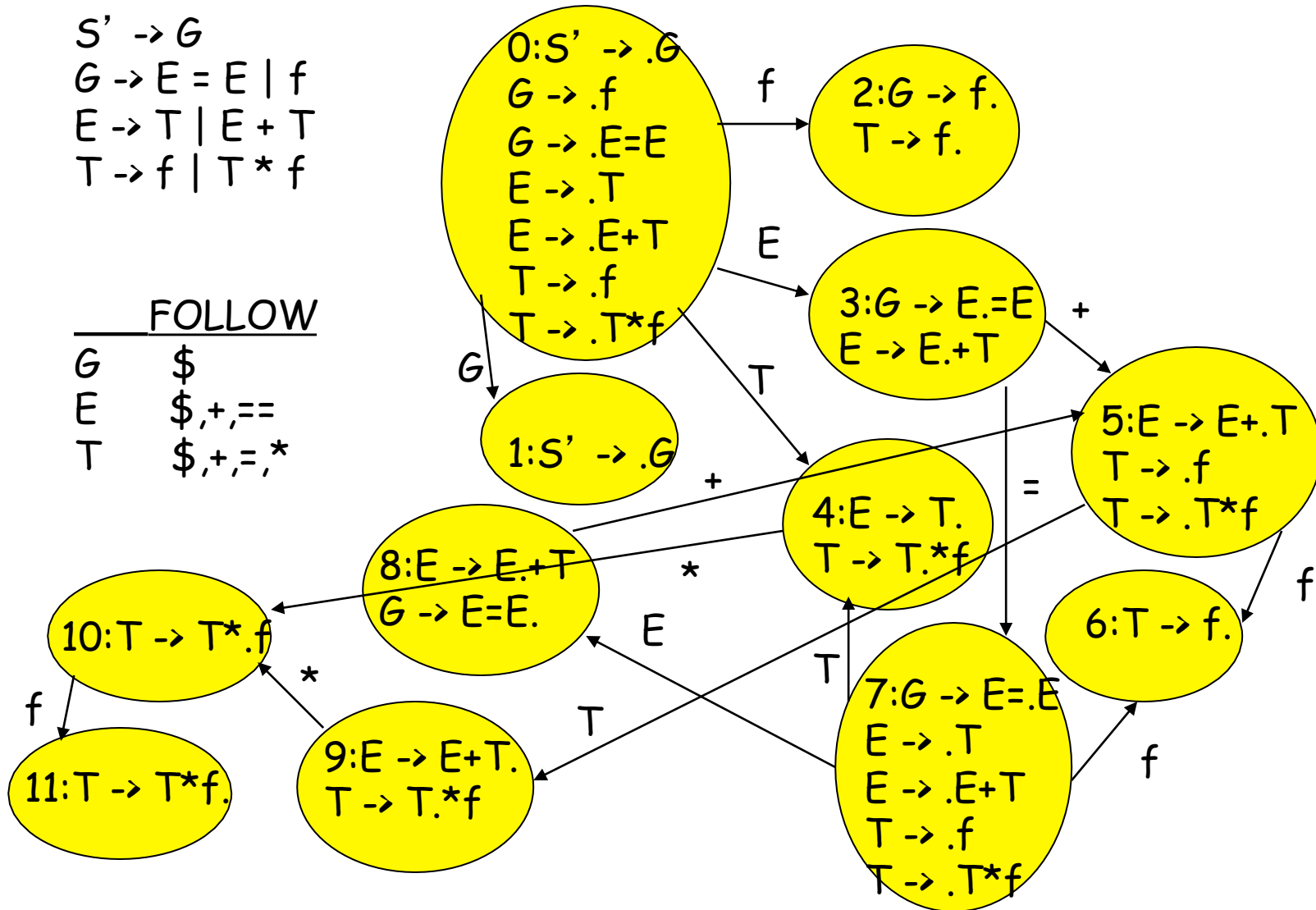
	Follow
S	\$
L	=, \$
R	=, \$



Another Example - not SLR(1)

$S' \rightarrow G$
 $G \rightarrow E = E \mid f$
 $E \rightarrow T \mid E + T$
 $T \rightarrow f \mid T * f$

	FOLLOW
G	\$
E	\$, +, =
T	\$, +, =, *



LR(1) Parser (for same grammar)

