Compiler Design

SLR Parsing

LR(0) conflicts

$S' \rightarrow T$

$T \rightarrow F$

$T \rightarrow T * F$

$T \rightarrow id$

$F \rightarrow id \mid (T)$

$F \rightarrow id = T ;$

1: $F \rightarrow id \bullet$

$F \rightarrow id \bullet = T$

Shift/reduce conflict

1: $F \rightarrow id \bullet$

$T \rightarrow id \bullet$

Reduce/Reduce conflict

Need more lookahead: **SLR(1)**

Example First/Follow

$S \rightarrow cAa$

$A \rightarrow cB \mid B$

$B \rightarrow bcB \mid \varepsilon$

$First(A) = \{b, c, \varepsilon\}$ $\quad$ $Follow(A) = \{a\}$

$First(B) = \{b, \varepsilon\}$ $\quad$ $Follow(B) = \{a\}$

$First(S) = \{c\}$ $\quad$ $Follow(S) = \{\$\}$

SLR(1) : Simple LR(1) Parsing

$$S' \rightarrow T$$
$$T \rightarrow F \mid T * F \mid C (T)$$
$$F \rightarrow id \mid id ++ \mid (T)$$
$$C \rightarrow id$$

What can the next symbol be when we reduce **F → id ?**

Follow(F) = { *,), $ }

S' → T → F → id

S' → T → T * F → T*id → F*id → id* id

S'→ T → C(T) → C(F) → C(id)

When we reduce F → id, the **top of the stack will contain:** id

And the **next input symbol will be** one of: $, *, or)

These symbols tell the parser whether it should shift or reduce during parsing.

The top of the stack will be id, and the next input symbol will be either **$**, *****, or **)**

SLR(1) : Simple LR(1) Parsing

$$S' \rightarrow T$$
$$T \rightarrow F \mid T * F \mid C\ (\ T\ )$$
$$F \rightarrow id \mid id\ ++ \mid (\ T\ )$$
$$C \rightarrow id$$

What can the next symbol be when we reduce **C → id ?** **Follow(C) = { (}**

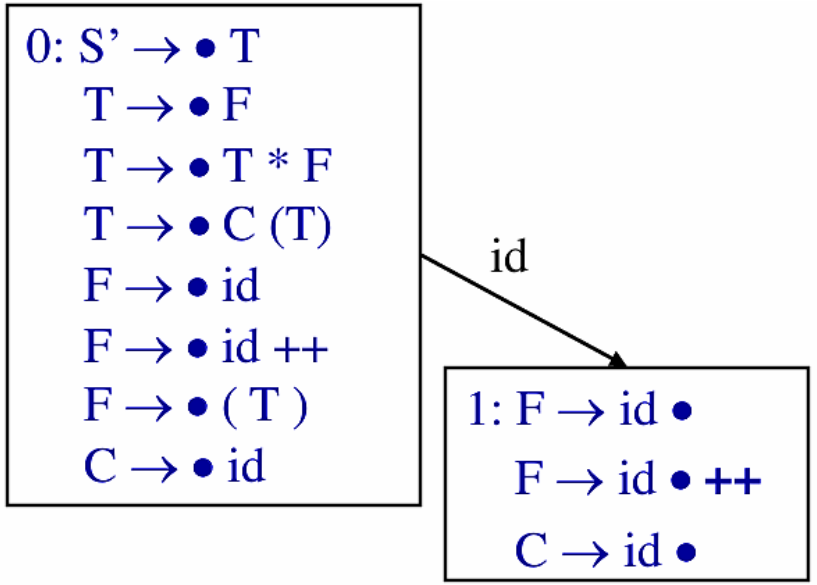
S' → T → C(T) → C(F) → C(id) → id (id)

When reducing C → id, the parser checks the FOLLOW(C) set to decide whether a reduction is valid based on the upcoming input symbol.

So, the valid lookahead symbol is: **'('**

SLR(1): Simple LR(1) Parsing

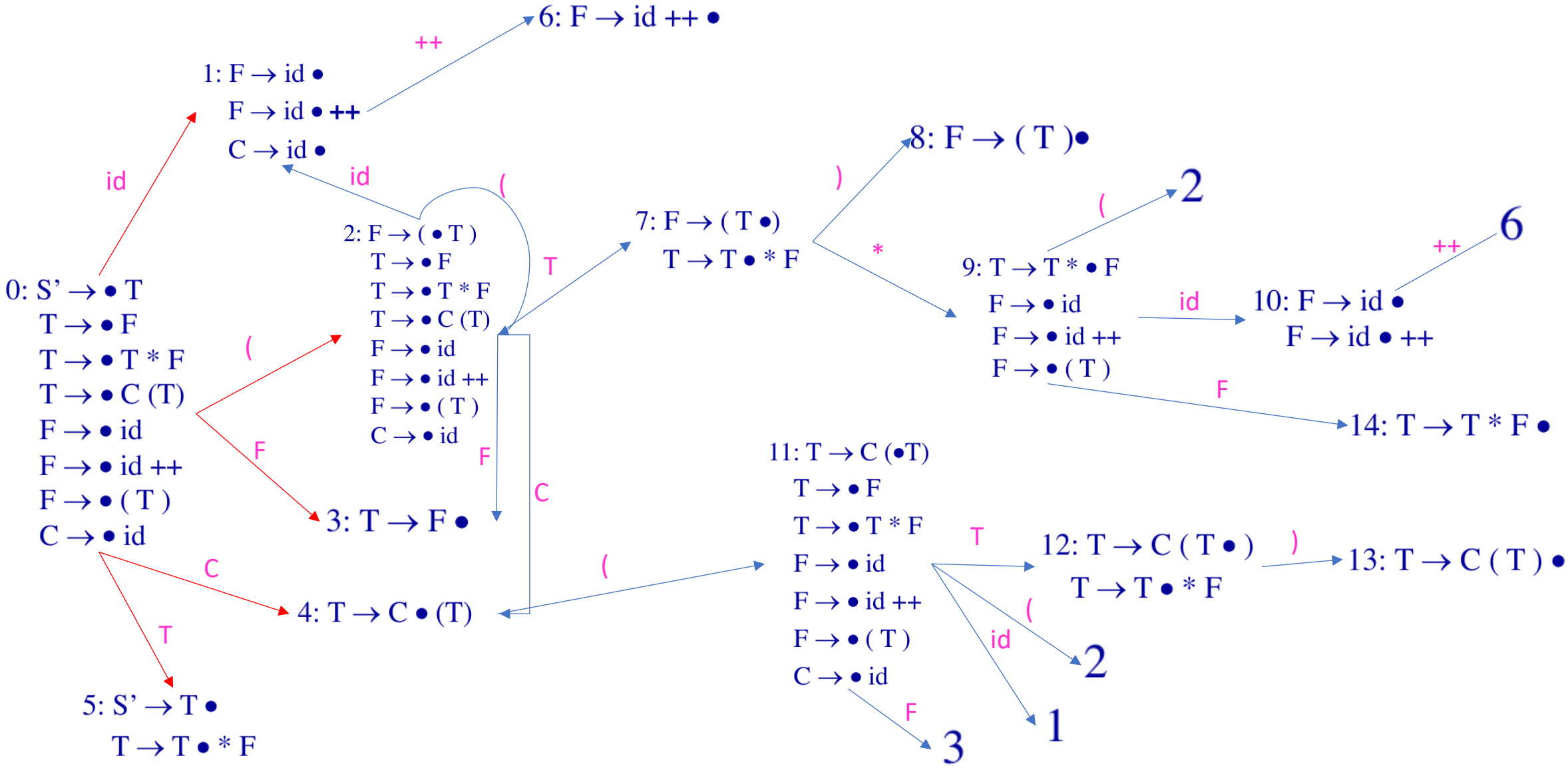
$S' \rightarrow T$
$T \rightarrow F \mid T * F \mid C\ (\ T\ )$
$F \rightarrow id \mid id\ ++ \mid (\ T\ )$
$C \rightarrow id$



Follow(F) = { *,), $ }
Follow(C) = { (}

action[1,*]= action[1,)] = action[1,$] = Reduce F → id
action[1,(] = Reduce C → id
action[1,++] = Shift

- It demonstrates how **the SLR(1) parser uses FOLLOW sets to decide reduction** actions.
- Shows conflict resolution: the same lookahead symbol can trigger either **reduce** or **shift**, depending on context.
- Provides an example of building states and table entries for grammar with id, ++, *, and parentheses



In SLR, → these FOLLOW sets are used for reduction decisions in the **ACTION table**.

SLR Parsing

Productions	
1	T → F
2	T → T*F
3	T → C(T)
4	F → id
5	F → id ++
6	F → (T)
7	C → id

FIRST(T) = { id, (}
FIRST(F) = { id, (}
FIRST(C) = { id }

FOLLOW(T) = { $, *,) }
FOLLOW(F) = { $, *,) }
FOLLOW(C) = { (}

	*	(	)	id	++	$	T	F	C
0		S2		S1			5	3	4
1	R4	R7	R4		S2	R4			
2		S2		S1			7	3	4
3	R1		R1			R1			
4		S11							
5	S9					A			
6	R5		R5			R5			
7	S9		S8						
8	R6		R6			R6			
9		S2		S10				14	
10	R4		R4		S6	R4			
11		S2		S1			12	3	
12	S9		S13						
13	R3		R3			R3			
14	R2		R2			R2			

SLR Parsing

- Assume:
  - Stack contains α and next input is t
  - DFA on input α terminates in state s

So, parser is in **state s** with **lookahead t**.

- Reduce by $X \rightarrow \beta$ if
  - s contains item $X \rightarrow \beta \bullet$
  - $t \in Follow(X)$

We can safely reduce since the RHS is complete, and lookahead matches a valid continuation

If there are still conflicts under these rules, grammar is not SLR(1)

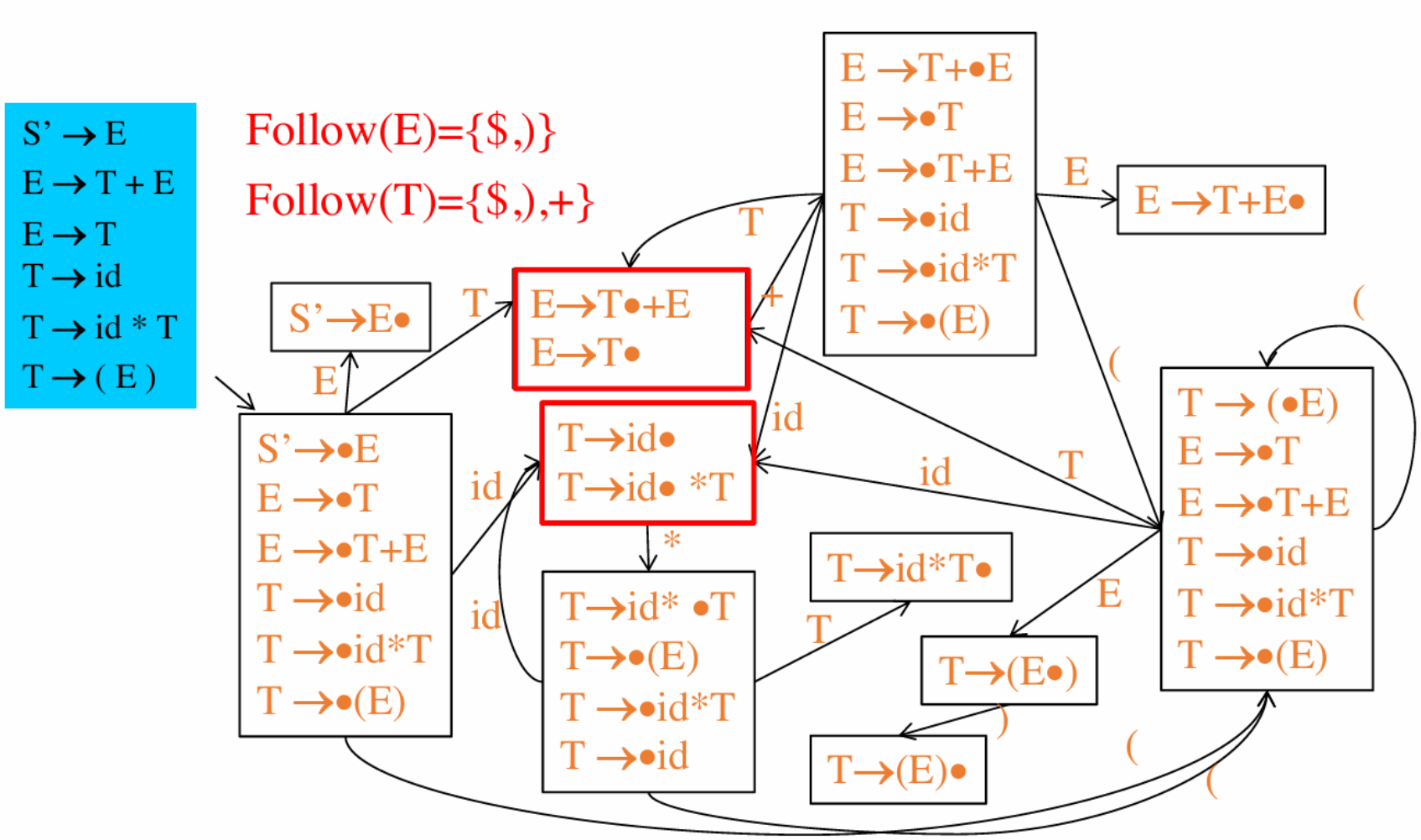
- Shift if
  - s contains item $X \rightarrow \beta \bullet t\omega$
  - If $Y \rightarrow \beta \bullet$ is in s then t cannot be in Follow(Y) for any Y

Dot before the next input symbol.

This prevents ambiguity between “reduce” and “shift.”

SLR Parsing

- **Example: State With E**
  - If lookahead is in FOLLOW(E) (**$ or)**), then **reduce E→T**.
  - If lookahead is **+**, then **shift** to parse more.



The slide is a **state machine for SLR(1) parsing** of arithmetic grammar, showing how parser shifts and reduces based on items + FOLLOW sets

SLR Parsing

- Let M be the finite-state automaton for viable prefixes of G
- Let $|x_1 \ldots x_n\$$ be initial configuration
- Repeat until configuration is $S|\$$
  - Let $\alpha| \omega$ be current configuration
  - Run M on current stack α
  - If M rejects α, report parsing error
    - Stack α is not a viable prefix
  - If M accepts α with items I, let a be the next input
    - Shift $[X \rightarrow \beta \bullet a\ \gamma] \in I$
    - Reduce if $[X \rightarrow \beta \bullet] \in I$ and $a \in Follow(X)$
    - Report parsing error if neither applies

If there is any conflict in the last step (more than two valid actions), grammar is not SLR(1)

- SLR parser uses an automaton for viable prefixes.
- At each step, it decides **shift** or **reduce** using FOLLOW sets.
- Parsing stops successfully when configuration becomes S|$.
- If conflicts exist → grammar not SLR(1).

Trace id*id

$S' \rightarrow E$

$E \rightarrow T + E$

$E \rightarrow T$

$T \rightarrow id$

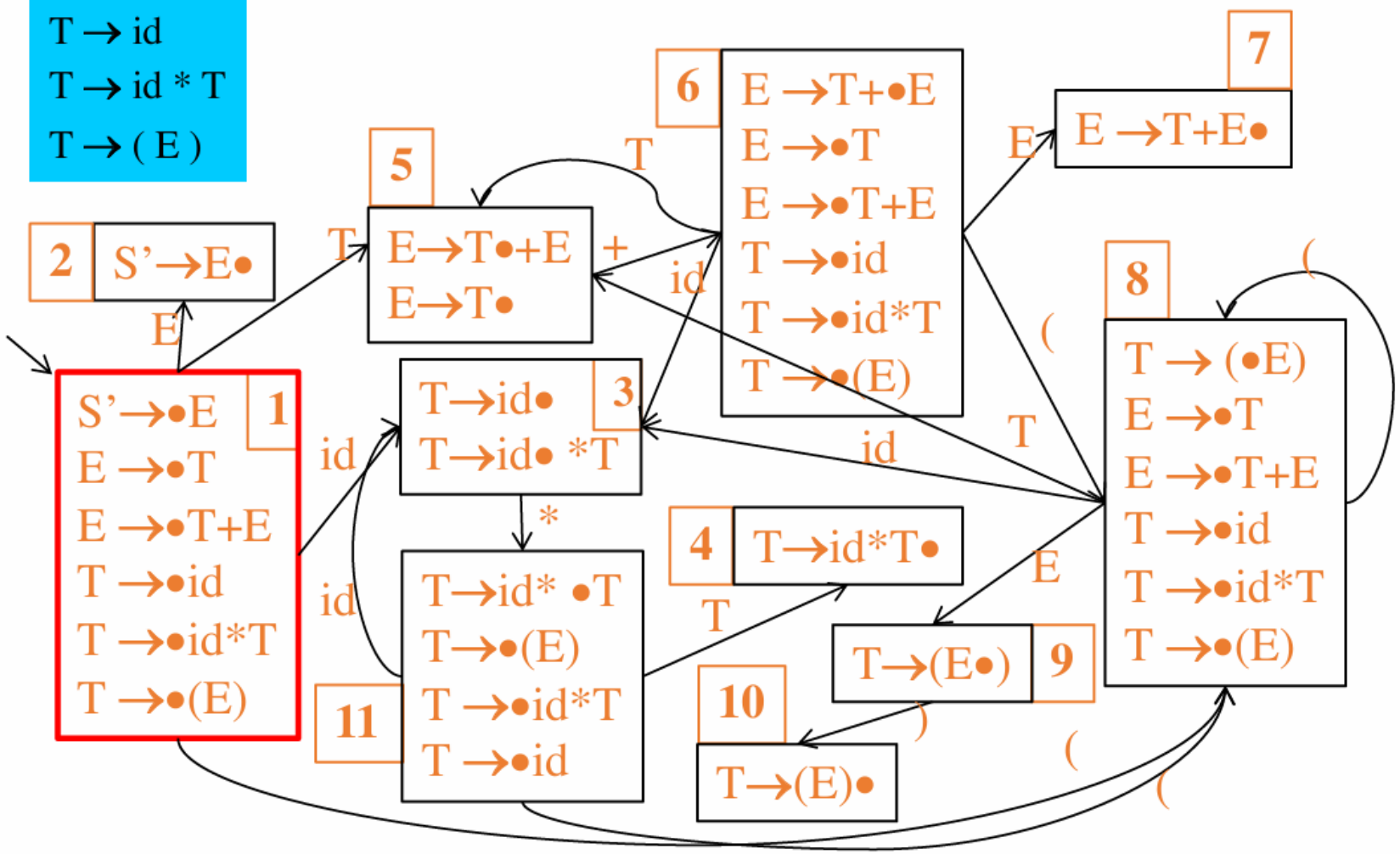
$T \rightarrow id * T$

$T \rightarrow (E)$

Input	Stack	Action
| id * id $		

S' → E
E → T + E
E → T
T → id
T → id * T
T → (E)

id * id $



Trace id*id

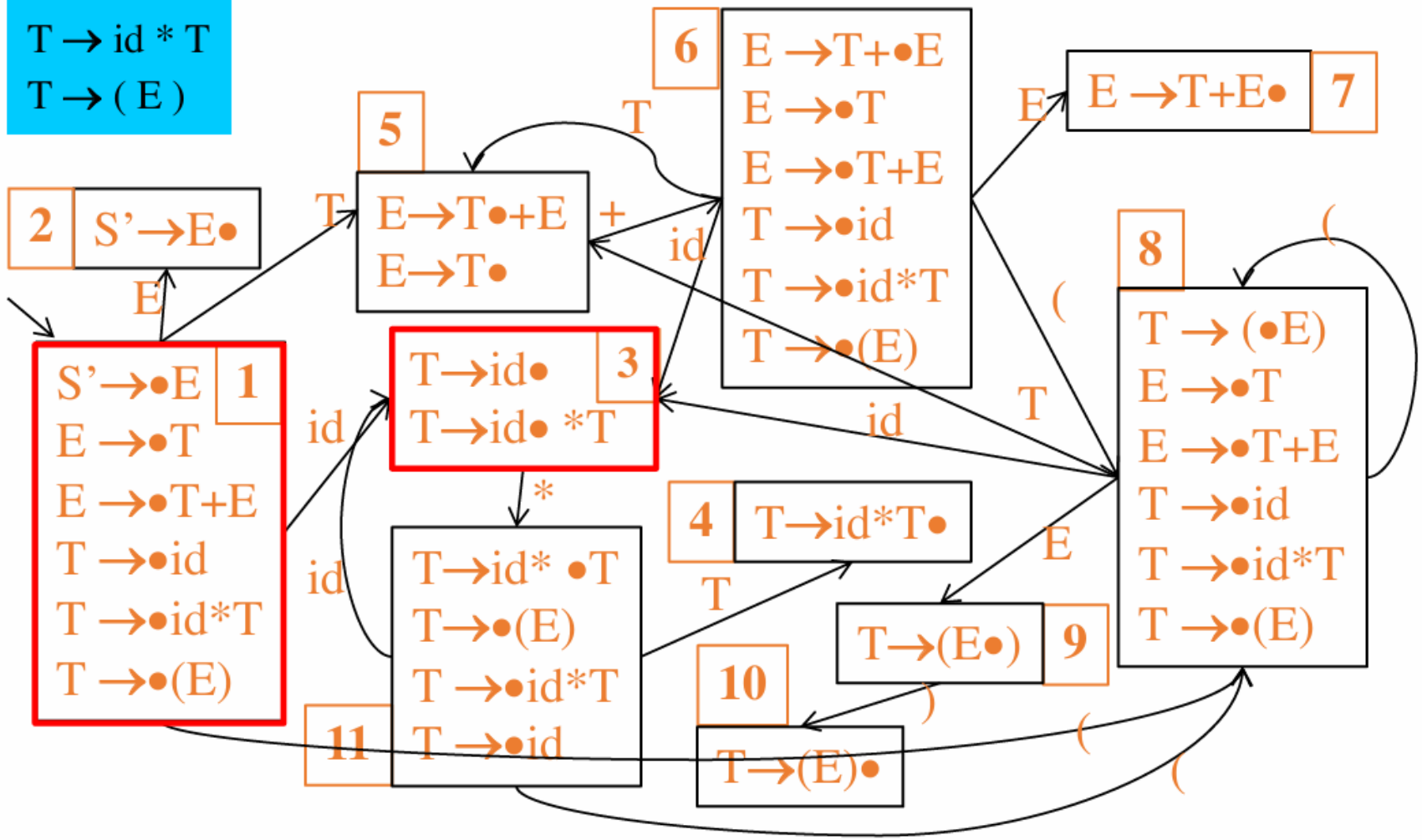
$S' \rightarrow E$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow id$
$T \rightarrow id * T$
$T \rightarrow (E)$

Input	Stack	Action
| id * id $ id | * id $	1	Shift 3

S' → E
E → T + E
E → T
T → id
T → id * T
T → (E)

id | * id $

Follow(T)={$,),+}



Trace id*id

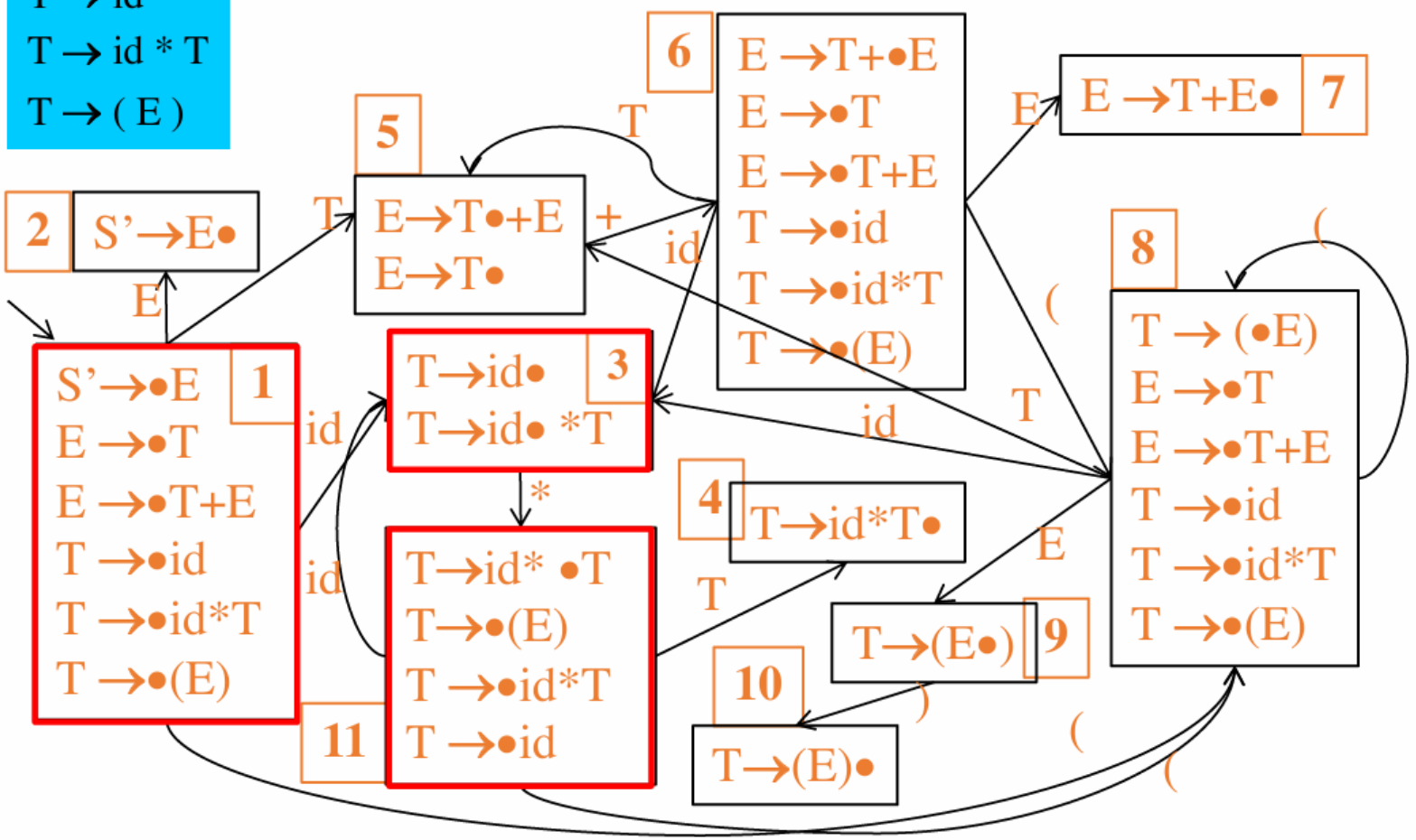
Trace id*id

$S' \rightarrow E$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow id$
$T \rightarrow id * T$
$T \rightarrow (E)$

Input	Stack	Action
| id * id $	1	Shift 3
id | * id $	1 3 * ∉ Follow(T)	Shift 11
id * | id $		

S' → E
E → T + E
E → T
T → id
T → id * T
T → (E)

id * | id $



Trace id*id

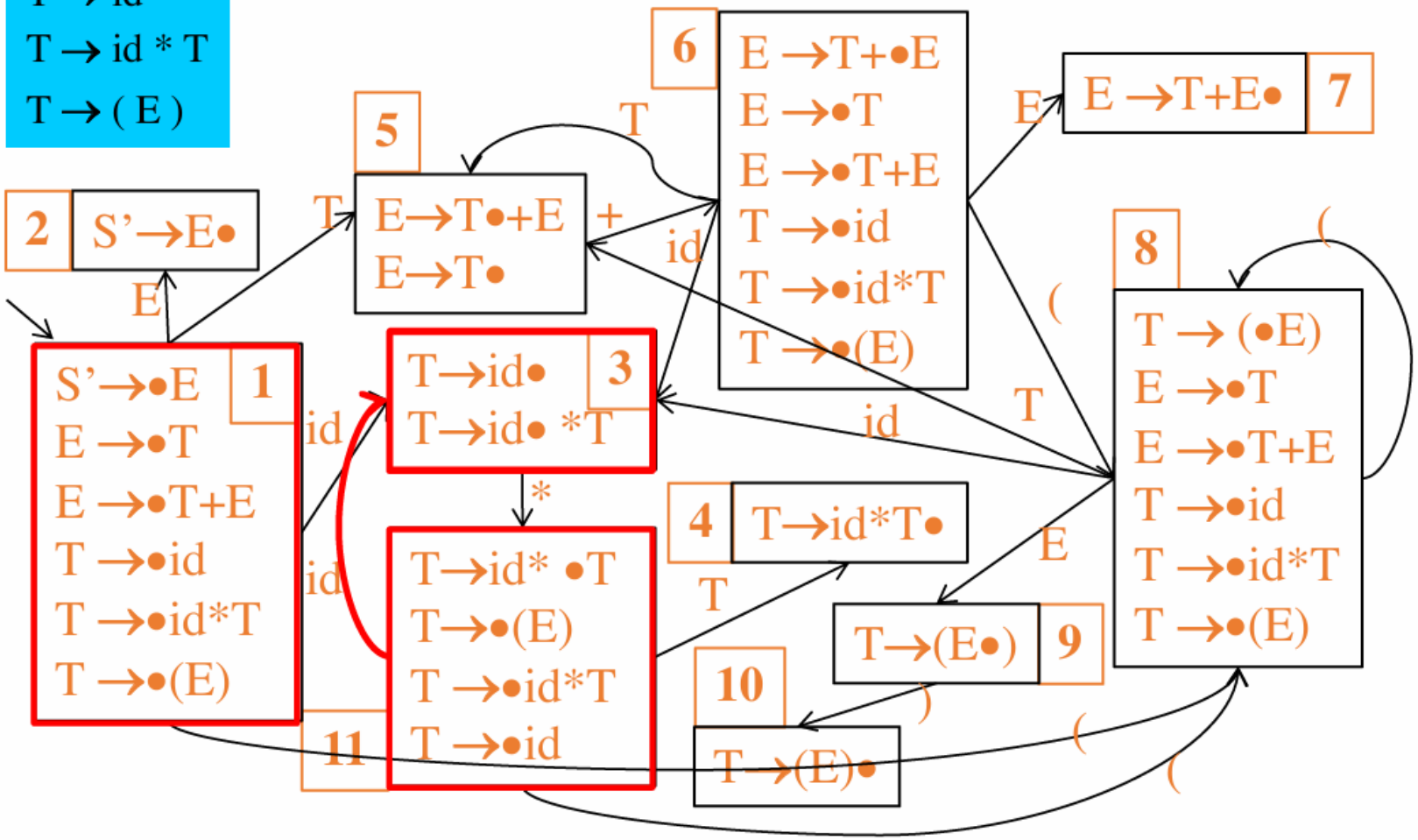
$S' \rightarrow E$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow id$
$T \rightarrow id * T$
$T \rightarrow (E)$

Input	Stack	Action
| id * id $	1	Shift 3
id | * id $	1 3 * ∉ Follow(T)	Shift 11
id * | id $	1 3 11	Shift 3
id * id | $		

S' → E
E → T + E
E → T
T → id
T → id * T
T → (E)

id * id | $

Follow(T)={$,),+}



Trace id*id

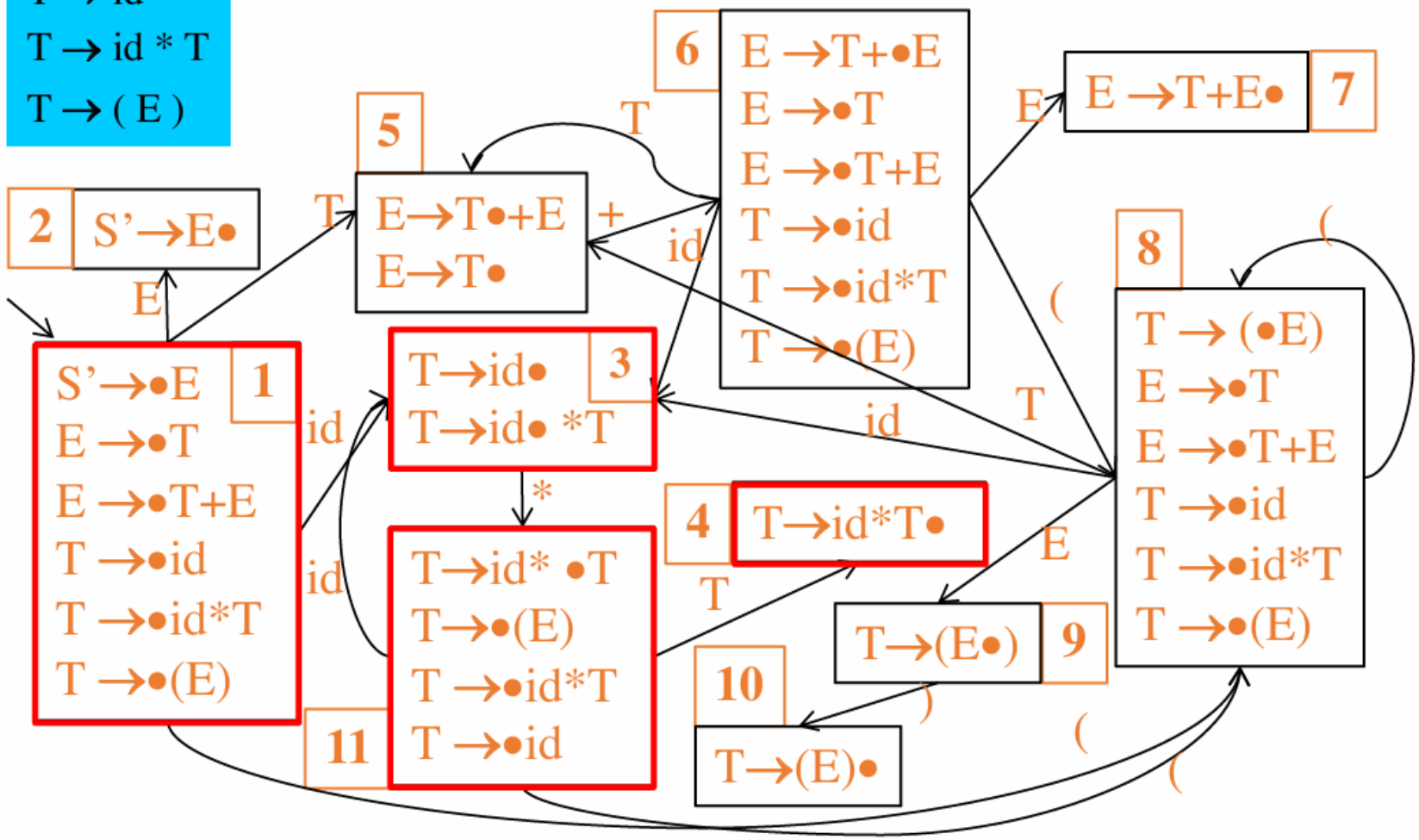
$S' \rightarrow E$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow id$
$T \rightarrow id * T$
$T \rightarrow (E)$

Input	Stack	Action
| id * id $	1	Shift
id | * id $	1 3 * ∉ Follow(T)	Shift
id * | id $	1 3 11	Shift
id * id | $	1 3 11 3 $ ∈ Follow(T)	Reduce T → id
id * T | $		

S' → E
E → T + E
E → T
T → id
T → id * T
T → (E)

id * T | $

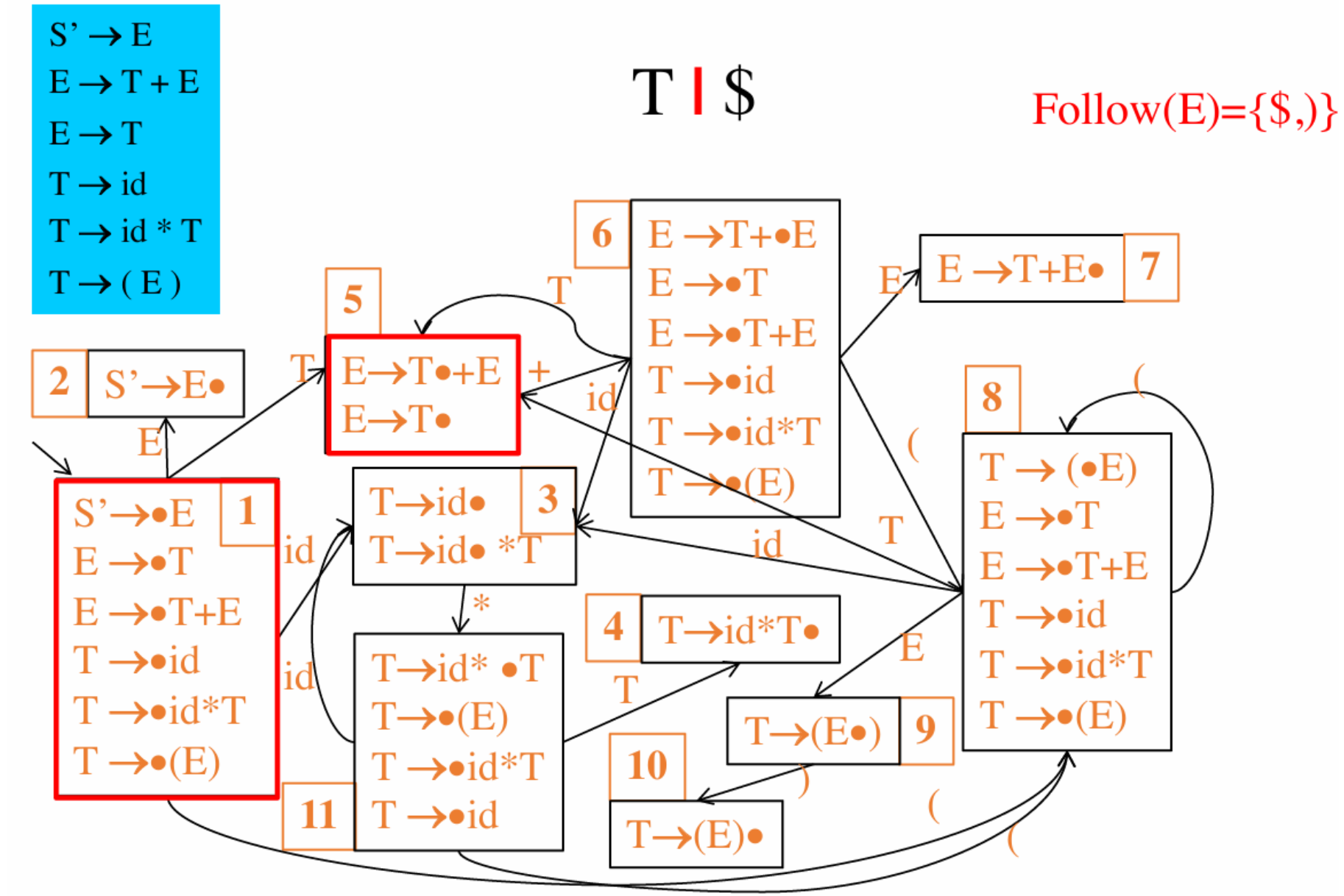
Follow(T)={$,),+}



Trace id*id

$S' \rightarrow E$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow id$
$T \rightarrow id * T$
$T \rightarrow (E)$

Input	Stack	Action
| id * id $	1	Shift
id | * id $	1 3 $\quad * \notin$ Follow(T)	Shift
id * | id $	1 3 11	Shift
id * id | $	1 3 11 3 $\quad \$ \in$ Follow(T)	Reduce T $\rightarrow$ id
id * T | $	1 3 11 4 $\quad \$ \in$ Follow(T)	Reduce T $\rightarrow$ id * T
T | $		


S' → E
E → T + E
E → T
T → id
T → id * T
T → (E)
T | $
Follow(E)={$,)}
1
S'→•E
E →•T
E →•T+E
T →•id
T →•id*T
T →•(E)
2
S'→E•
3
T→id•
T→id• *T
4
T→id*T•
5
E→T•+E
E→T•
6
E →T+•E
E →•T
E →•T+E
T →•id
T →•id*T
T →•(E)
7
E →T+E•
8
T → (•E)
E →•T
E →•T+E
T →•id
T →•id*T
T →•(E)
9
T→(E•)
10
T→(E)•
11
T→id* •T
T→•(E)
T →•id*T
T →•id
E
T
id
+
*
(
)

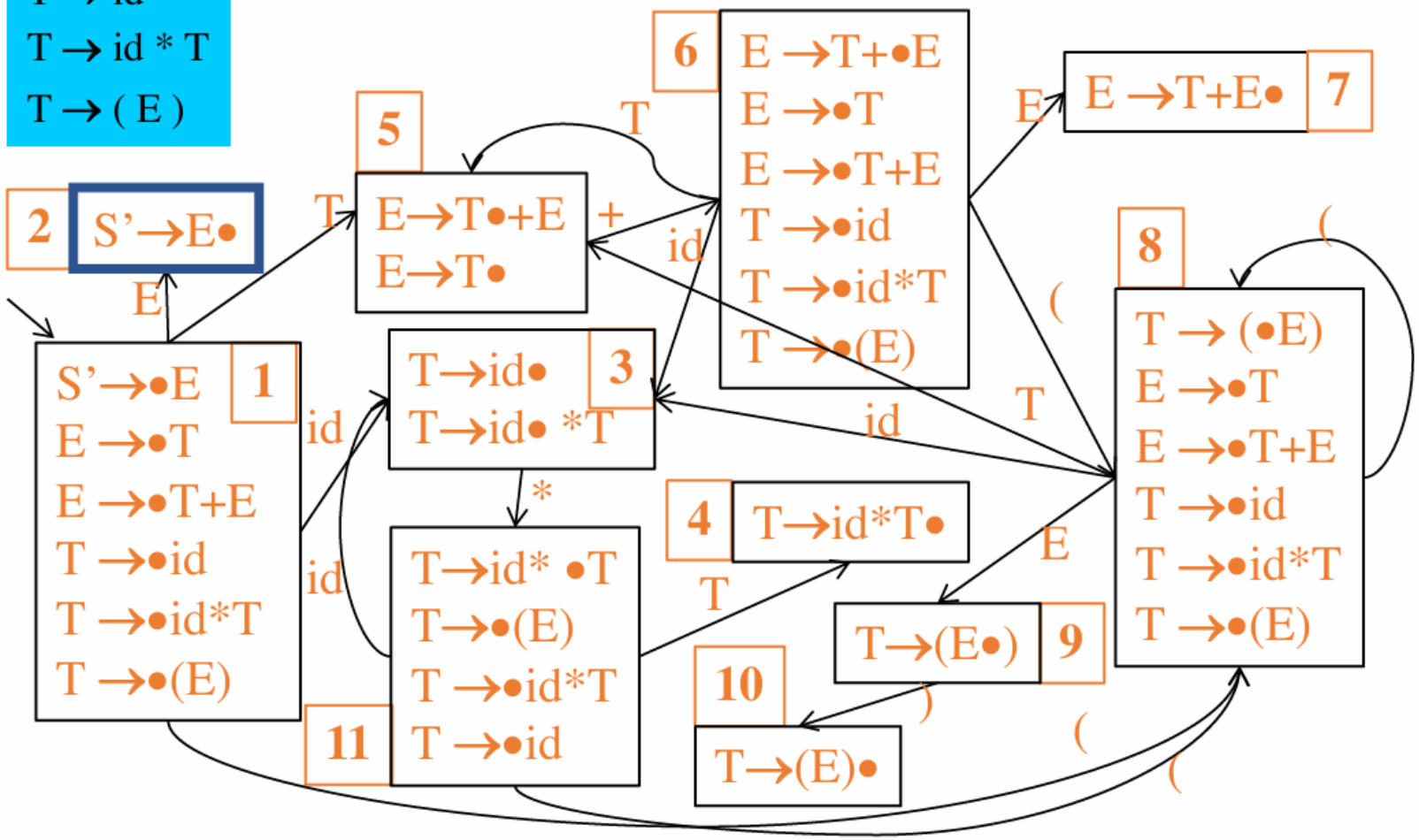
Trace id*id

$S' \rightarrow E$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow id$
$T \rightarrow id * T$
$T \rightarrow (E)$

Input	Stack	Action
| id * id $	1	Shift
id | * id $	1 3 $*\notin$ Follow(T)	Shift
id * | id $	1 3 11	Shift
id * id | $	1 3 11 3 $\$ \in$ Follow(T)	Reduce T $\rightarrow$ id
id * T | $	1 3 11 4 $\$ \in$ Follow(T)	Reduce T $\rightarrow$ id * T
T | $	1 5 $\$ \in$ Follow(T)	Reduce E$\rightarrow$T
E | $		

S' → E
E → T + E
E → T
T → id
T → id * T
T → (E)

E | $

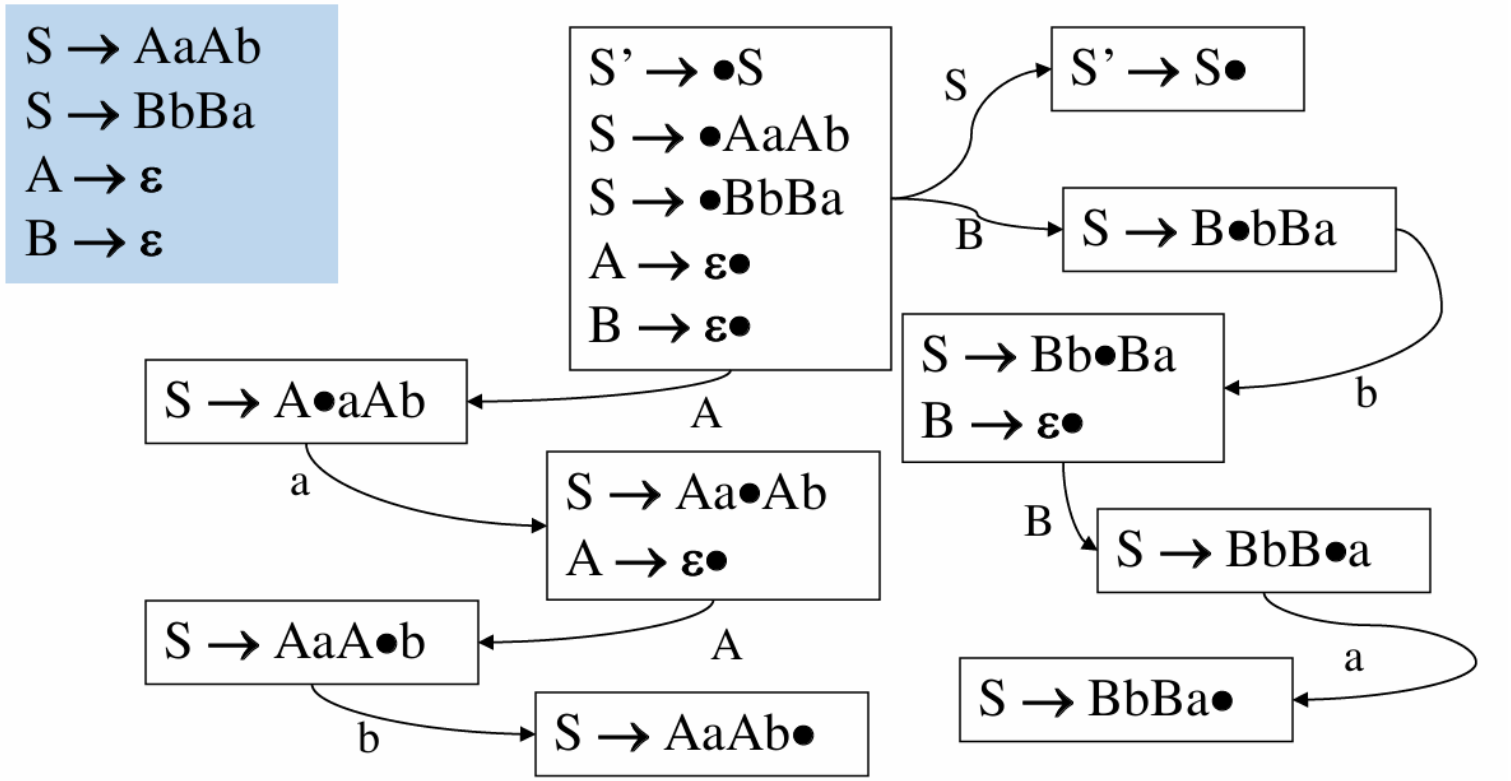


Trace id*id

$S' \rightarrow E$
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow id$
$T \rightarrow id * T$
$T \rightarrow (E)$

Input	Stack	Action
| id * id $	1	Shift
id | * id $	1 3 $* \notin$ Follow(T)	Shift
id * | id $	1 3 11	Shift
id * id | $	1 3 11 3 $\$ \in$ Follow(T)	Reduce T → id
id * T | $	1 3 11 4 $\$ \in$ Follow(T)	Reduce T → id * T
T | $	1 5 $\$ \in$ Follow(T)	Reduce E→T
E | $	1 2 $\$ \in$ Follow(E)	Accept

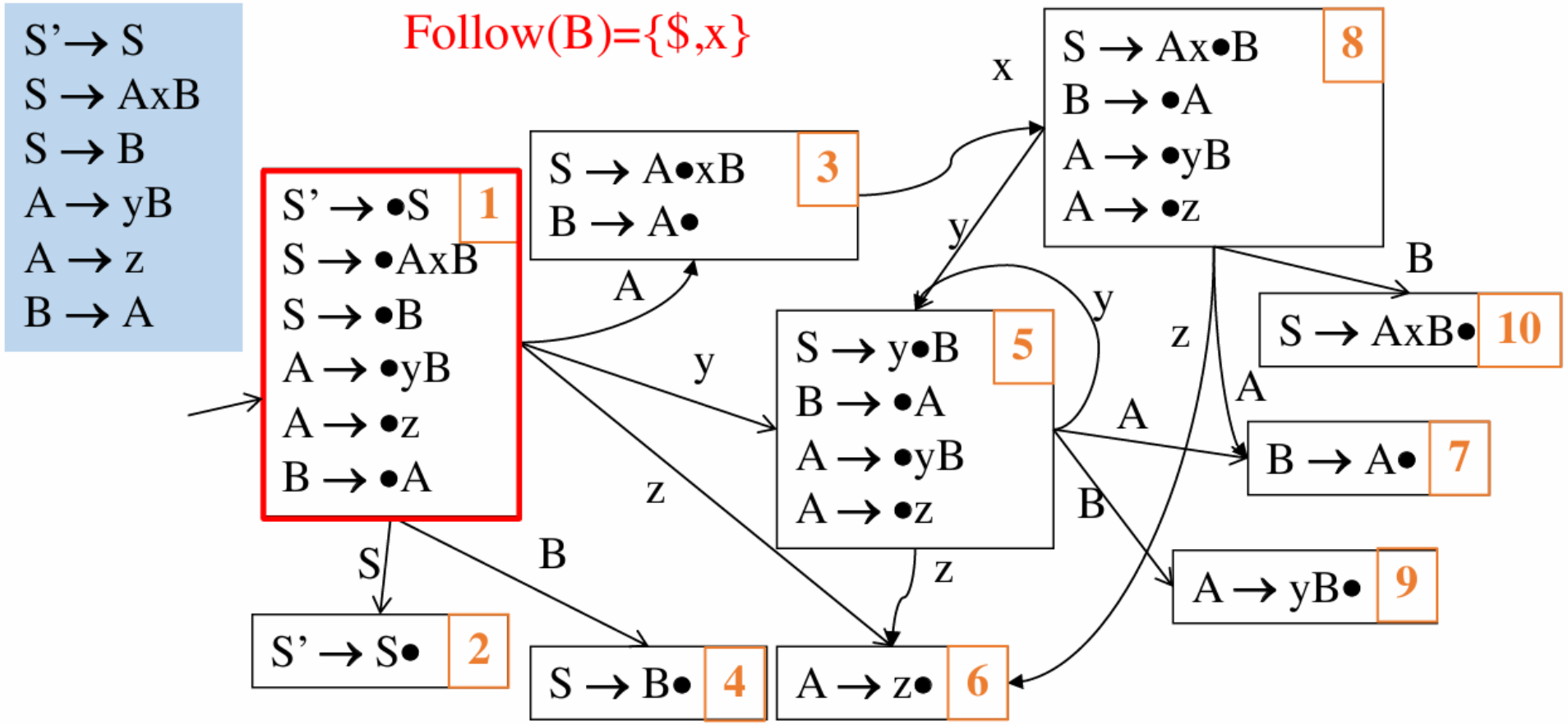
Is this grammar SLR(1)?



- In the initial item set (shown in the diagram), both completed items A → ε· and B → ε· appear in the *same state*.
- Because Follow(A) = Follow(B) = {a,b}, for lookahead a (or b) **both** reductions A → ε and B → ε are valid in that same state.
- That yields a **reduce–reduce conflict** (two different reductions enabled by the same lookahead), so the SLR(1) decision rules are not sufficient — the parsing table would be conflicted.

Conclusion: Grammar is **not SLR(1)**.

Is this grammar SLR(1)?



SLR Parsing Table

Grammar is not SLR

0) S' → S
1) S → AxB
2) S → B
3) A → yB
4) A → z
5) B → A

	x	y	z	$	S	A	B
1		S5	S6		2	3	4
2				ACC!			
3	S8,R5			R5			
4				R2			
5		S5	S6			7	9
6	R4			R4			
7	R5			R5			
8		S5	S6			7	10
9	R3			R3			
10				R1			