Chapter 6

상향식 파싱 알고리즘 - **Part II**

Table of Contents

SLR(1) Parsing

- ***Simple*** **LR(1)**
  - DFA of sets of LR(0) items
  - uses the ***next token*** in the input string
    - *consults* the input token *before* a shift
      - 해당 transition 존재 여부를 확인
        - ◆ 진짜 그런 기호가 입력에 있니?
  - uses the **`Follow`** set of a nonterminal
    - reduction 실행 여부를 결정
      - ◆ 이렇게 reduction 하는 게 정말 맞는 거니?

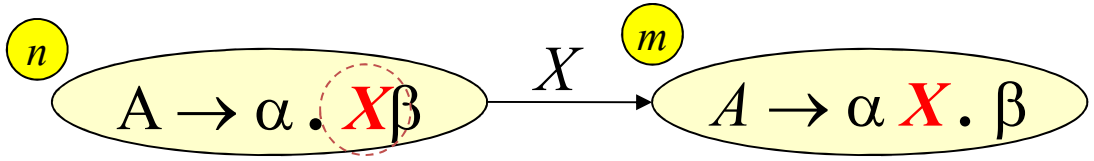
- 이렇게만 해도 LR(0)에 비해 엄청난 power-up!!!

SLR(1) Parsing Algorithm (1/3)

■ 현재 상태 *n* 이

1. ***shift*** item $A \rightarrow \alpha .X\beta$ 을 갖고 있는 경우

- **X**가 *terminal* 이고, ***입력에서 읽어 온 token 도 X 이면***,
  - ***shift* action** : *input token* **X**를 stack에 push
  - $A \rightarrow \alpha X . \beta$ 의 item을 포함하는 상태 번호(*m*)를 push



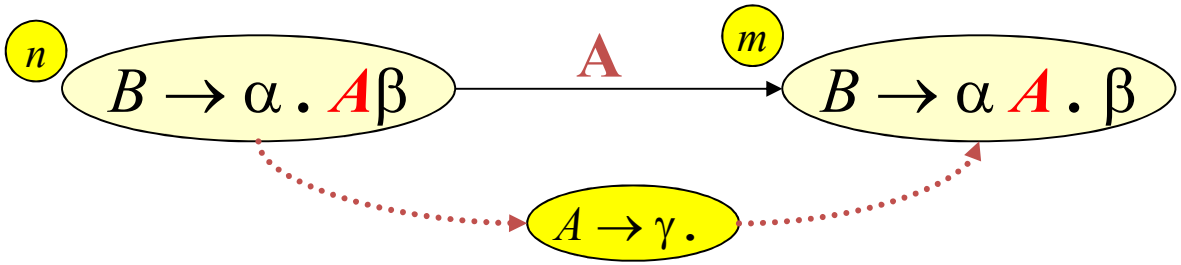
Parsing Stack	Input
$ . . . n	**X** . . . $
$ . . . n **X** **m**	. . . $

SLR(1) Parsing Algorithm (2/3)

■ 현재 상태 *n* 이

2. complete item $A \rightarrow \gamma$. 을 갖고 있는 경우

- *입력에서 읽어 온 token이* ***Follow(A)에 속하면***,
  - ***reduce* action**: reduction by $A \rightarrow \gamma$
    - stack에서 기호(γ) 와 상태 번호를 함께 제거(*pop*)
    - stack에 nonterminal ***A***를 push
    - $B \rightarrow \alpha$. $A\beta$ 에서 reduction직후의 item $B \rightarrow \alpha A$. β 를 포함하는 상태 번호를 push



SLR(1) Parsing Algorithm (3/3)

■ 현재 상태 *n* 이

2. complete item ***S′*** → ***S.*** 을 갖고 있는 경우

- 입력 버퍼가 비어 있으면(*LOOKAHEAD='$'*)

➔ Parsing은 여기서 멈춤

➔ The input sentence is syntactically correct for a given grammar.

SLR(1) Grammar

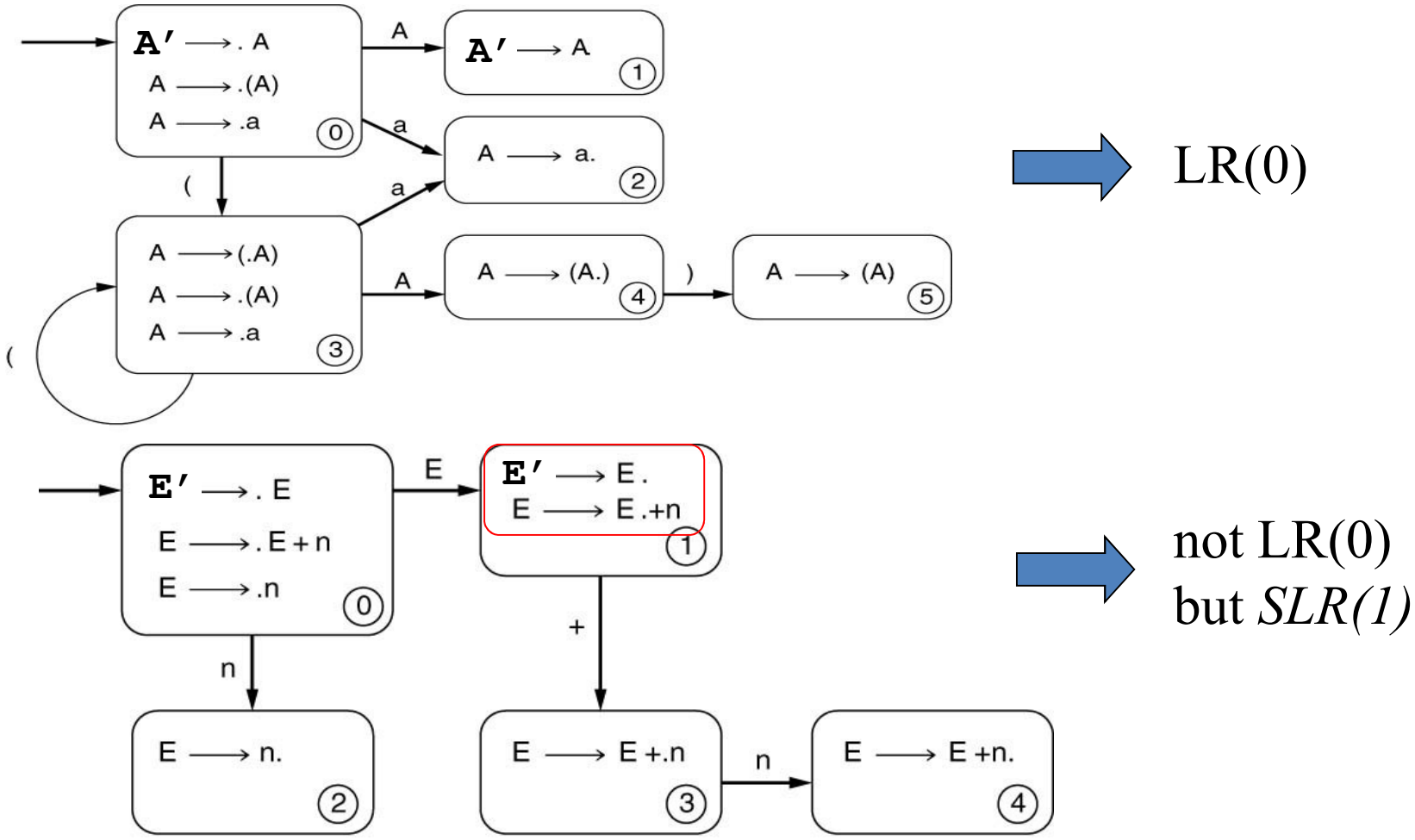
- **A grammar is *SLR(1) grammar* *if and only if* the following two conditions are satisfied:**

for any state `s`,

1. **no *shift-reduce conflict***
   - For any *shift* item $A \rightarrow \alpha.X\beta$ in `s` with X a *terminal*,
     - there is ***no*** *complete* item $B \rightarrow \gamma$. in `s` **with X** in **Follow(*B*).**

2. **no *reduce-reduce conflict***
   - For any two *complete* items $A \rightarrow \alpha$. and $B \rightarrow \beta$. in `s`,
     - **Follow(*A*) ∩ Follow(*B*)** is *empty*.

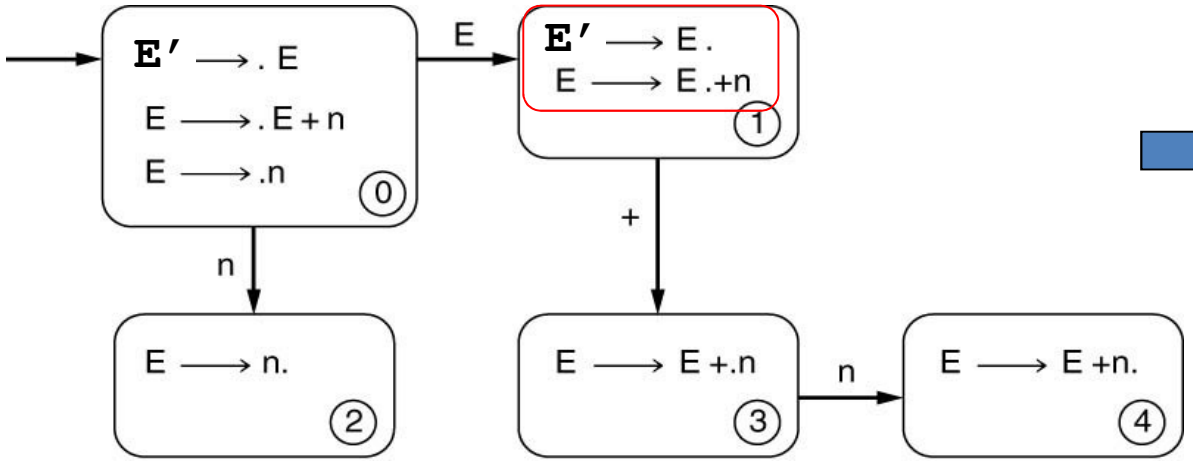
What's the difference?



LR(0)

not LR(0)
but *SLR(1)*

What's the difference?



not LR(0)
but *SLR(1)*

Why?

shift item E → E .+ n

reduce item E′ → E .

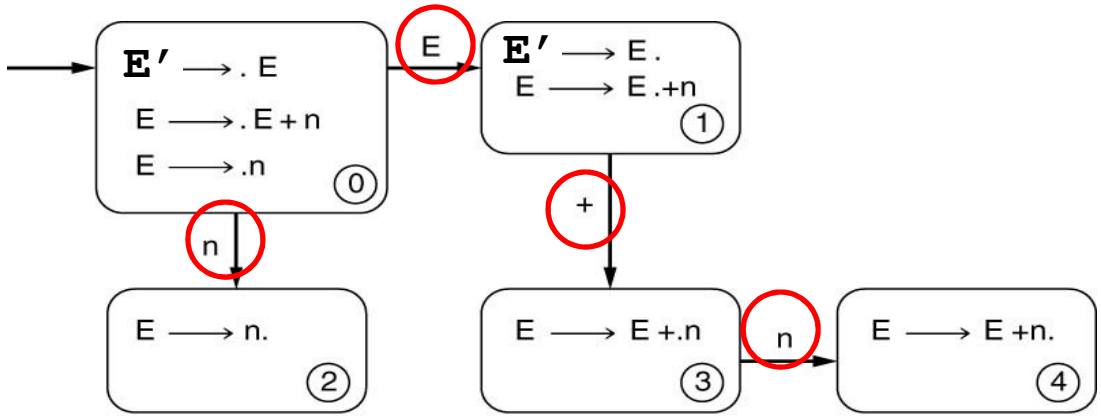
SLR(1) Parsing Table (1/2)

- **SLR(1) parsing table 구성**

기호 / 상태	ACTION 표	GOTO 표
	$V_T \cup \{\$\}$	V_N
0 1 2 :	shift reduce accept error	상태 번호

- **4 semantic actions**
  1. **Shift** : ACTION[S_m, a_i] = shift S
  2. **Reduce** : ACTION[S_m, a_i] = reduce by A $\rightarrow \alpha$
  3. **Accept** : ACTION[S_m, a_i] = accept
  4. **Syntax error** : ACTION[S_m, a_i] = error (No action defined!)

SLR(1) Parsing Table for Example 10

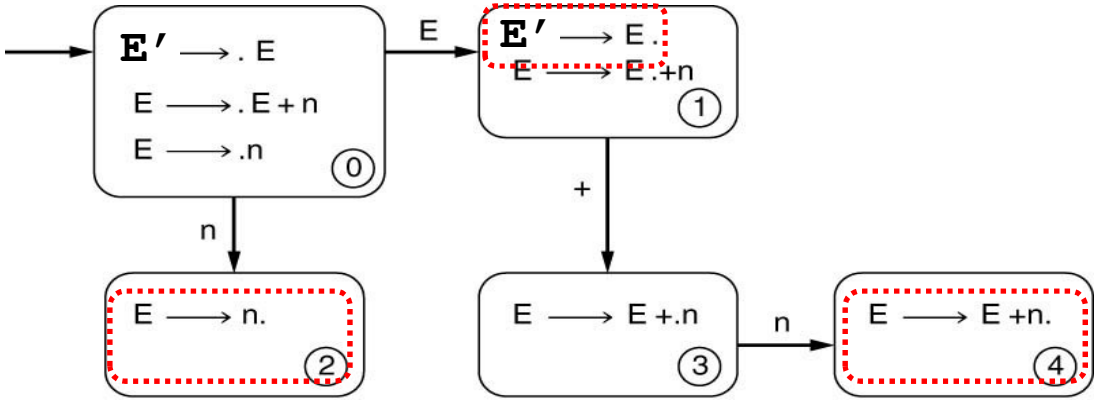



```
E′ → E
E  → E + n | n
```

State	input			goto
	n	+	$	E
0	s2			1
1		s3		
2				
3	s4			
4				

shift 와 *reduce* action이 같은 상태에 함께 포함될 수 있음
shift action + 상태번호 ➔ s2

SLR(1) Parsing Table for Example 10



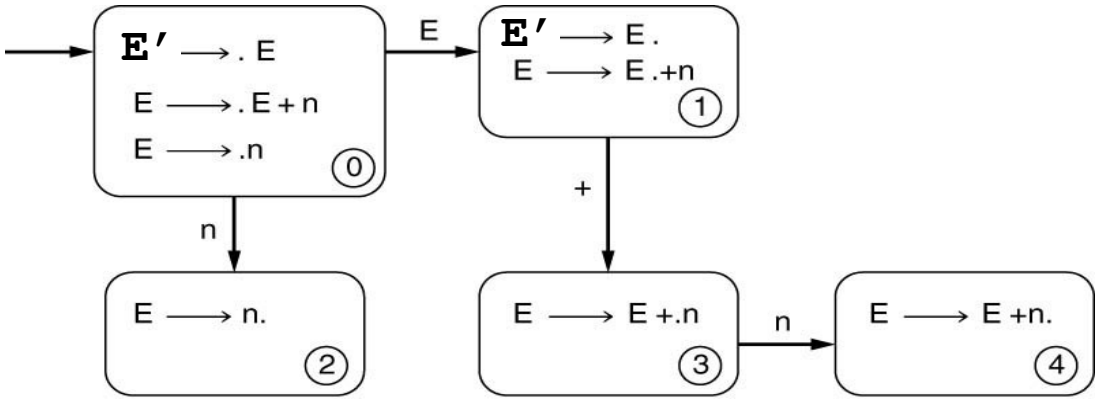
E′ → E
E → E + n | n

FOLLOW(E′) = {$}
FOLLOW(E) = {+, $}

State	input			goto
	n	+	$	E
0				1
1			accept	
2		r(E → n)	r(E → n)	
3				
4		r(E → E+n)	r(E → E+n)	

reduce action + *rule*

SLR(1) Parsing Table for Example 10



State	input			goto
	n	+	$	E
0	s2			1
1		s3	accept	
2		r(E → n)	r(E → n)	
3	s4			
4		r(E → E+n)	r(E → E+n)	

What was changed?

State	Input			Goto
	n	**+**	**$**	**E**
0	s2			1
1		s3	accept	
2		r(E→n)	r(E→n)	
3	s4			
4		r(E→E+n)	r(E→E+n)	

SLR(1)

LR(0)

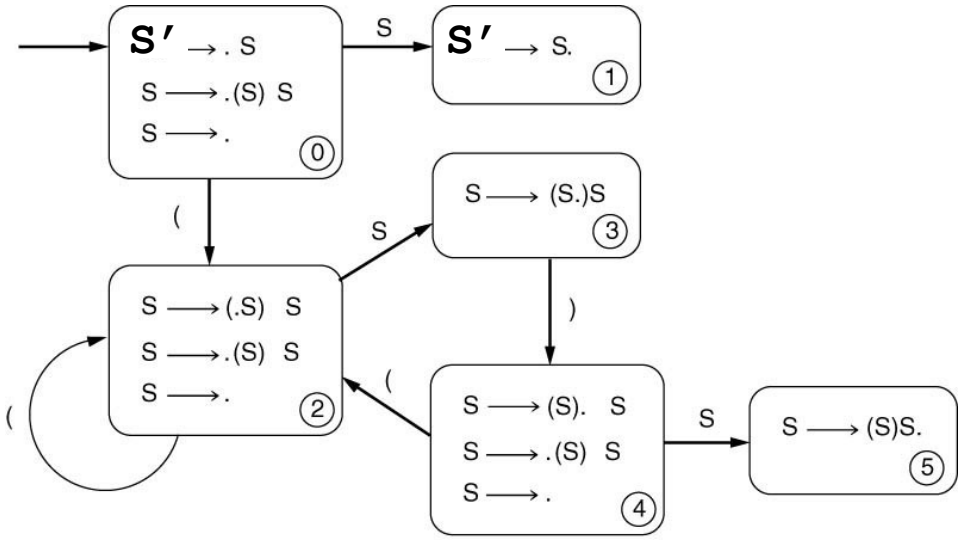
State	Action	Rule	Input			Goto
			(	**a**	**)**	**A**
0	shift		3	2		1
1	reduce	A′→A				
2	reduce	A →a				
3	shift		3	2		4
4	shift				5	
5	reduce	A→(A)				

Parsing Actions for Example 10

State	Input			Goto
	n	**+**	**$**	**E**
0	s2			1
1		s3	accept	
2		r(E→n)	r(E→n)	
3	s4			
4		r(E→E+n)	r(E→E+n)	

	Parsing Stack	Input	Action
1	$0	**n+n+n**$	shift 2
2	$0n2	+n+n$	reduce by E→n
3	$0E1	+n+n$	shift 3
4	$0E1+3	n+n$	shift 4
5	$0E1+3n4	+n$	reduce E→E+n
6	$0E1	+n$	shift 3
7	$0E1+3	n$	shift 4
8	$0E1+3n4	$	reduce by E→E+n
9	$0E1	$	accept

SLR(1) Parsing Table for Example 7



S′ → S
S → (S) S | ε

FOLLOW(S′) = {**$**}
FOLLOW(S)= {**)**, **$**}

State	Input			Goto
	(	)	$	*S*
0	s2	r(*S*→ε)	r(*S*→ε)	1
1			**accept**	
2	s2	r(*S*→ε)	r(*S*→ε)	3
3		s4		
4	s2	r(*S*→ε)	r(*S*→ε)	5
5		r(*S*→(*S*)*S*)	r(*S*→(*S*)*S*)	

Parsing Actions for Example 7

State	Input			Goto
	(	)	$	S
0	s2	r($S\rightarrow\varepsilon$)	r($S\rightarrow\varepsilon$)	1
1			**accept**	
2	s2	r($S\rightarrow\varepsilon$)	r($S\rightarrow\varepsilon$)	3
3		s4		
4	s2	r($S\rightarrow\varepsilon$)	r($S\rightarrow\varepsilon$)	5
5		r($S\rightarrow(S)\,S$)	r($S\rightarrow(S)\,S$)	

	Parsing Stack	Input	Action
1	$0	**()()**$	shift 2
2	$0(2	)()$	reduce $S\rightarrow\varepsilon$
3	$0(2S3	)()$	shift 4
4	$0(2S3)4	()$	shift 2
5	$0(2S3)4(2	)$	reduce $S\rightarrow\varepsilon$
6	$0(2S3)4(2S3	)$	shift 4
7	$0(2S3)4(2S3)4	$	reduce $S\rightarrow\varepsilon$
8	$0(2S3)4(2S3)4S5	$	reduce $S\rightarrow(S)S$
9	$0(2S3)4S5	$	reduce $S\rightarrow(S)S$
10	$0S1	$	accept

예 12 : SLR(1) parsing - C_0

- **C_0** : Canonical Collection of LR(0) items
- **[예] 1. E → E + T 2. E → T 3. T → T * F 4. T → F 5. F → (E) 6. F → id**
  - 확장 문법(augmented grammar)

0. **S′ → E** 1. E → E + T 2. E → T 3. T → T * F 4. T → F 5. F → (E) 6. F → id

I0 = CLOSURE([S′ →•E])
= {[S′ →•E], [E →•E + T], [E →•T], [T →• T * F], [T →•F], [F →•(E)], [F →•id]}

GOTO(I0, E) = I1 = {[S′ → E•], [E → E• + T]}

GOTO(I0, T) = I2 = {[E → T•], [T → T• * T]}

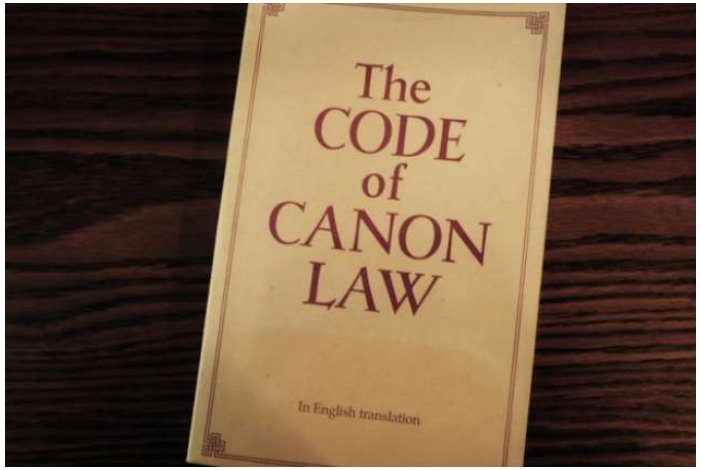
GOTO(I0, F) = I3 = {[T → F•]}

GOTO(I0, ( ) = I4 = CLOSURE([F → (•E)])
= {[F → (•E)], [E →•E + T], [E →•T], [T →•T * F], [T →•F], [F →•(E)], [F →•id]}

GOTO(I0, id) = I5 = {[F → id•]}

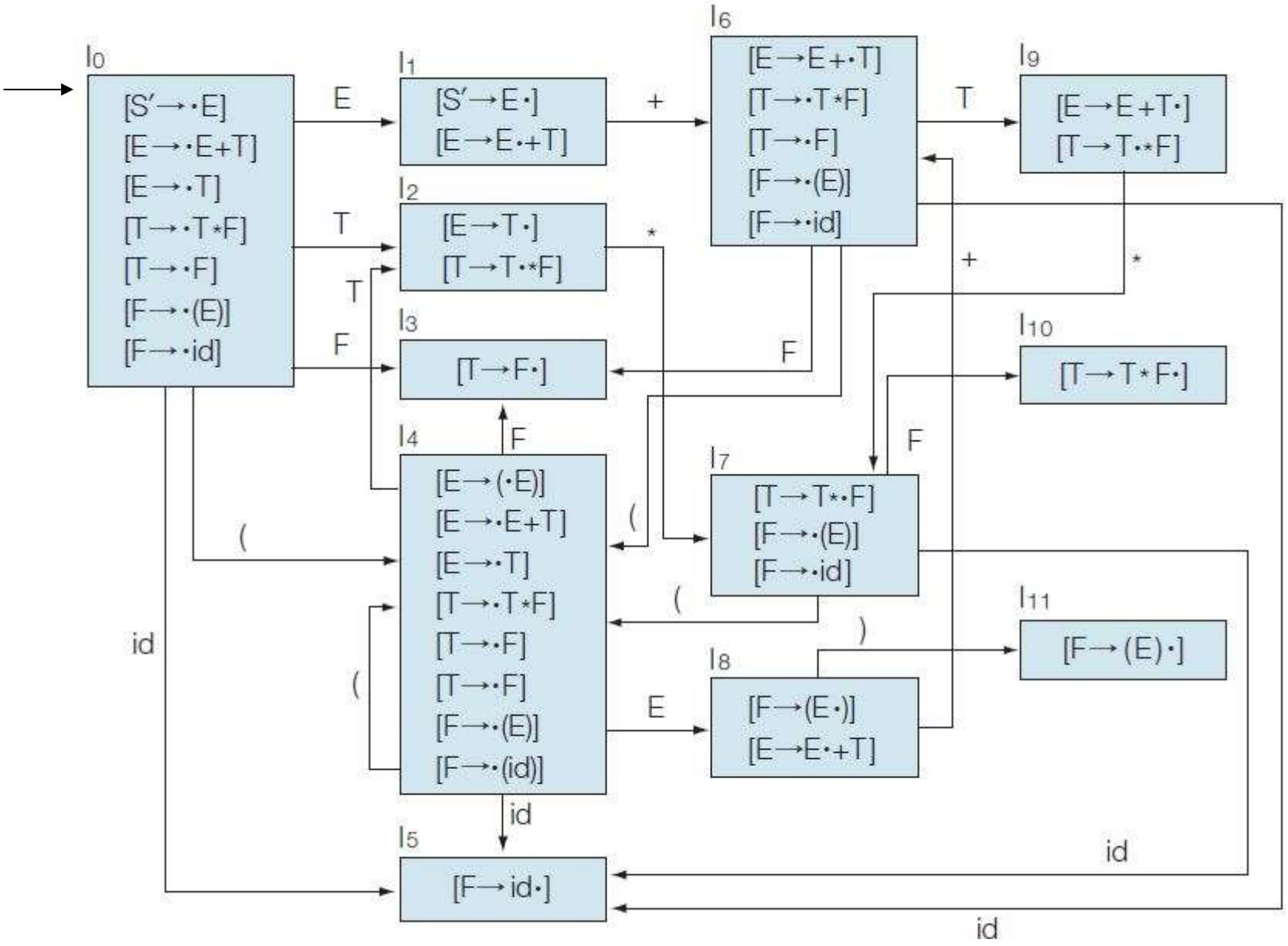
Canonical Collection of LR(0) items - C_0

- **Canonical** 은 무슨 뜻?
  - **Canon law** : 교회의 권위에 의해 만들어진 일련의 조례 및 규정



  - **Canonical Collection** ➔ 표준 모음(*Standard Collection*)

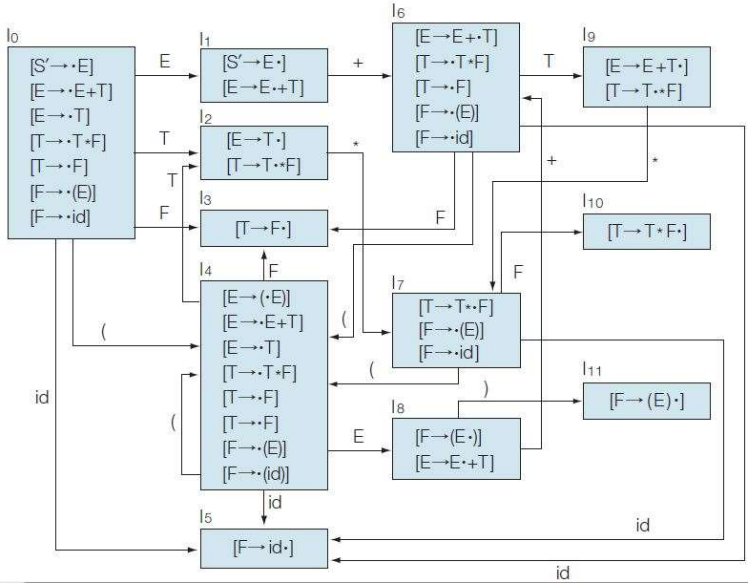
예 12 : SLR(1) parsing - DFA of LR(0) items



예 12 : SLR(1) parsing – parsing table

0. S′ → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

상태	구문 분석기 행동						GOTO 함수		
	id	+	*	(	)	$	E	T	F
0	s5			s4			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			



FOLLOW(E) = {$, +,) }
FOLLOW(T) = {*, +,), $}
FOLLOW(F) = {*, +,), $}

예 12: SLR(1) parsing – syntax analysis (1/2)

표 6-3 SLR 파싱표

상태	구문 분석기 행동						GOTO 함수		
	id	+	*	(	)	$	E	T	F
0	s5			s4			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			

단계	스택	입력 기호	구문 분석 내용
0	0	id * (id * id)$	이동 5
1	0id5	* (id * id)$	감축 6
2	0F	* (id * id)$	GOTO 3
3	0F3	* (id * id)$	감축 4
4	0T	* (id * id)$	GOTO 2
5	0T2	* (id * id)$	이동 7
6	0T2 * 7	(id * id)$	이동 4
7	0T2 * 7(4	id * id)$	이동 5
8	0T2 * 7(4id5	* id)$	감축 6
9	0T2 * 7(4F	* id)$	GOTO 3

예 12: SLR(1) parsing – syntax analysis (2/2)

표 6-3 SLR 파싱표

상태	구문 분석기 행동						GOTO 함수		
	id	+	*	(	)	$	E	T	F
0	s5			s4			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			

단계	스택	입력 기호	구문 분석 내용
10	0T2 * 7(4F3	* id)$	감축 4
11	0T2 * 7(4T	* id)$	GOTO 2
12	0T2 * 7(4T2	* id)$	이동 7
13	0T2 * 7(4T2 * 7	id)$	이동 5
14	0T2 * 7(4T2 * 7id5	)$	감축 6
15	0T2 * 7(4T2 * 7F	)$	GOTO 10
16	0T2 * 7(4T2 * 7F10	)$	감축 3
17	0T2 * 7(4T	)$	GOTO 2
18	0T2 * 7(4T2	)$	감축 2
19	0T2 * 7(4E	)$	GOTO 8
20	0T2 * 7(4E8	)$	이동 11
21	0T2 * 7(4E8)11	$	감축 5
22	0T2 * 7F	$	GOTO 10
23	0T2 * 7F10	$	감축 3
24	0T	$	GOTO 2
25	0T2	$	감축 2
26	0E	$	GOTO 1
27	0E1	$	수락

Disambiguating Rules for Parsing Conflicts

shift-reduce conflicts 해결 방법

- always ***prefer the shift*** over the reduce
  - incorporates the *most closely nested rule* for the dangling else ambiguity in **`if`**-statement
- 자연스러운 해결 방법
  - 문법에 모호함이 있더라도 이런 방식으로 해결 가능

reduce-reduce conflicts

- 대부분 문법 자체에 오류가 있는 경우 발생

예13

- **Grammar with *dangling-else* ambiguity**

  S → I | **other**

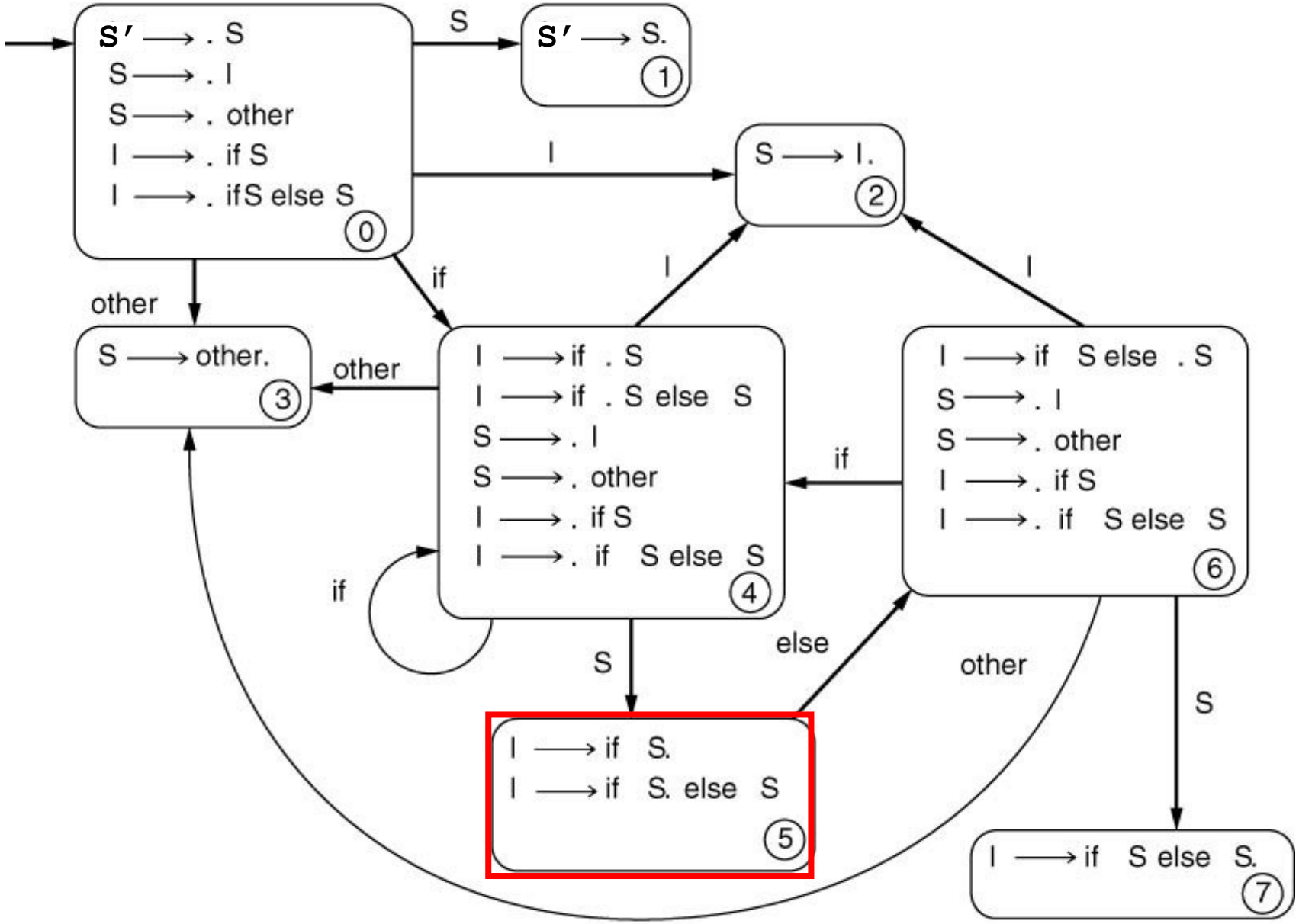
  I → **if** S | **if** S **else** S

- **DFA를 구성하면 parsing conflict이 발생하는 상태 존재**
  - *shift* item과 *reduce* item이 같은 상태에 포함됨

  I → **if** S . FOLLOW(I) = { **else**, $ }

  I → **if** S . **else** S 다음 토큰이 **else** 일 때 *shift* action

- **shift action을 우선 적용 → resolve the shift-reduce conflict**
  - **else**는 가장 가까운 **if**와 짝을 이룸(*most closely nested rule*)


S′ → . S
S → . I
S → . other
I → . if S
I → . if S else S
0
S
S′ → S.
1
I
S → I.
2
other
if
I
I
S → other.
3
other
I → if . S
I → if . S else S
S → . I
S → . other
I → . if S
I → . if S else S
4
if
if
I → if S else . S
S → . I
S → . other
I → . if S
I → . if S else S
6
if
S
else
other
S
I → if S.
I → if S. else S
5
I → if S else S.
7

예 13: SLR(1) Parsing Table

State	Input				Goto	
	if	**else**	**other**	**$**	*S*	*I*
0	s4		s3		1	2
1				accept		
2		r1		r1		
3		r2		r2		
4	r4		s3		5	2
5		**s6**		r3		
6	s4		s3		7	2
7		r4		r4		

(1) S → I
(2) S → other
(3) I → if S
(4) I → if S else S

Limits of SLR(1) Parsing Power (1/3)

- **"phony" problem (장난 전화)**
  - 실제 입력에서 절대로 나타날 수 없는 상황 때문에 conflict가 발생
    - caused by the *weakness* of the SLR(1) method
- **SLR(1) parsing을 적용할 수 없는 예**

stmt → *call-stmt* | *assign-stmt*

call-stmt → `identifier`

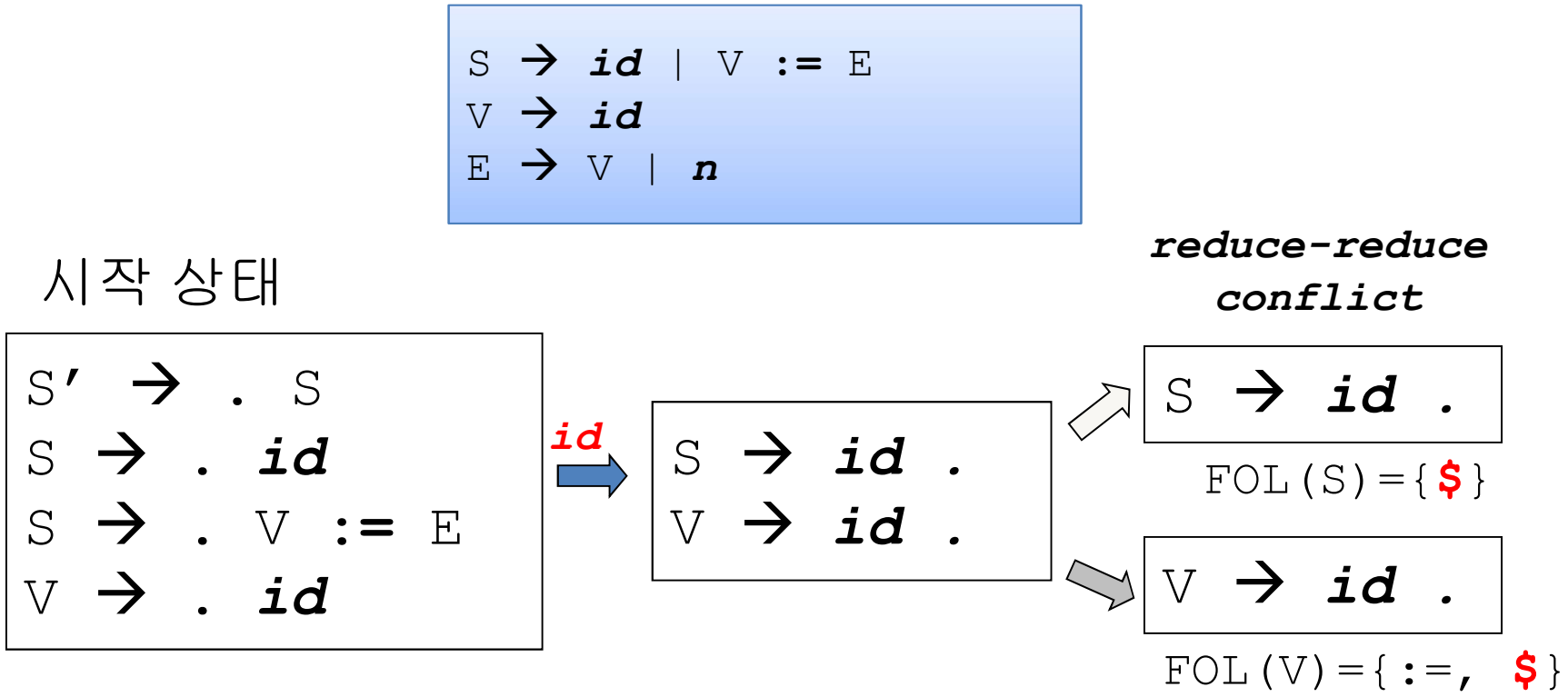
assign-stmt → *var* `:=` *exp*

var → *var* `[` *exp* `]` | `identifier`

exp → *var* | `number`

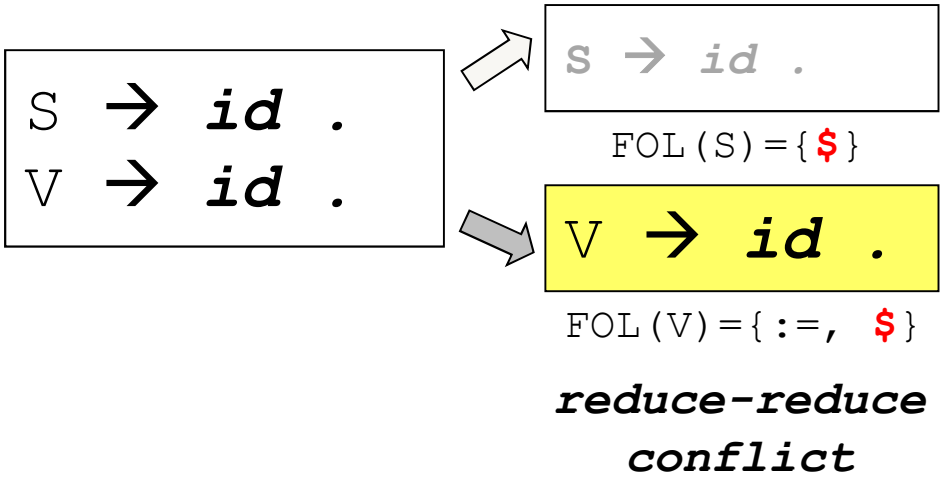
  - 호출문 (*call-stmt*)과 할당문(*assign-stmt*) 모두 **identifier**로 시작
    `:=` 를 읽어오기 전까지는 호출문인지 할당문인지 알 수 없음

Limits of SLR(1) Parsing Power (2/3)



Limits of SLR(1) Parsing Power (3/3)

```
S → id | V := E
V → id
E → V | n
```



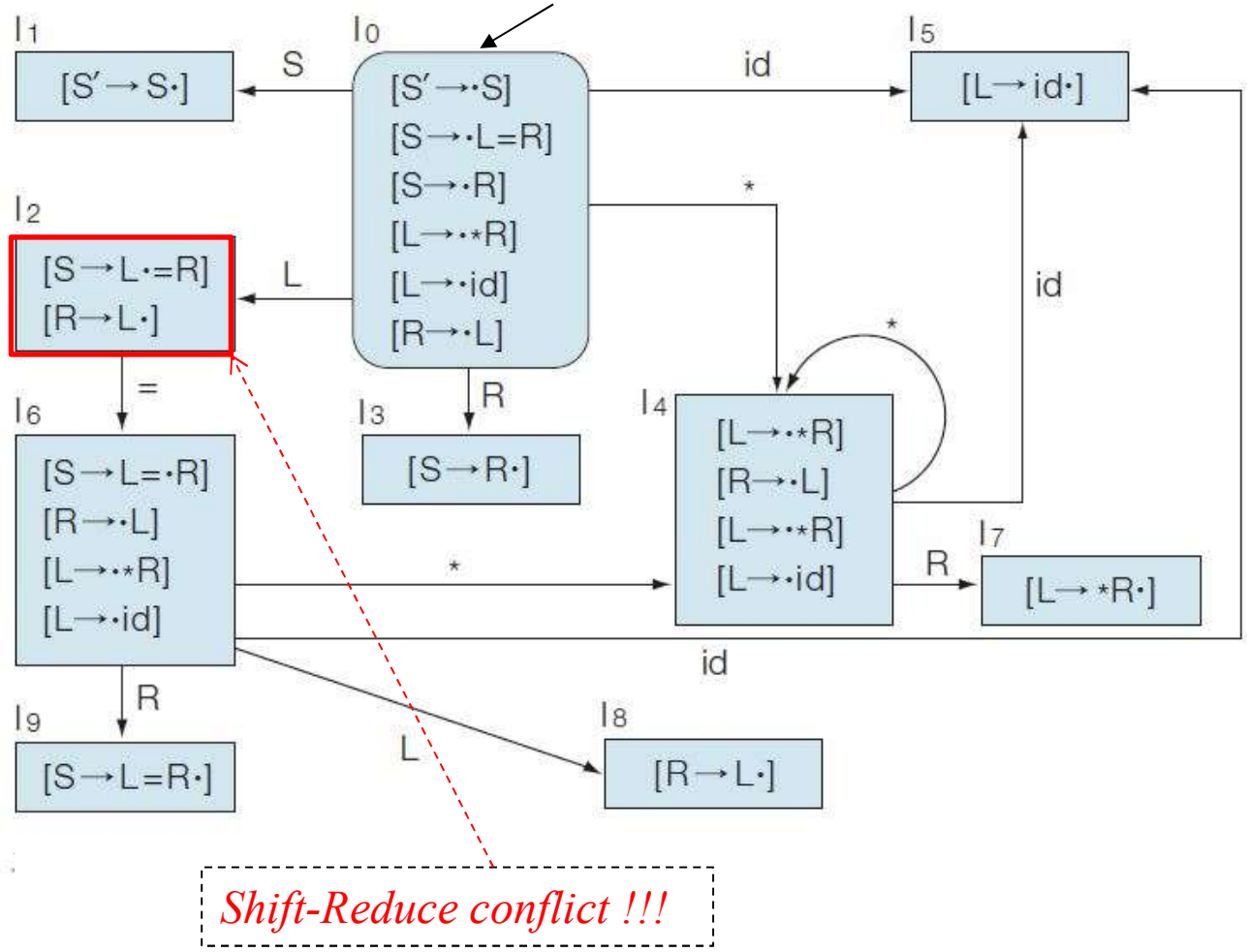
이 문장 구조에서 다음 token으로 $가 올 수 없음

$V := E$ 에서 $id := n$

절대 올 수 없는 기호 때문에 *parsing conflict* 발생

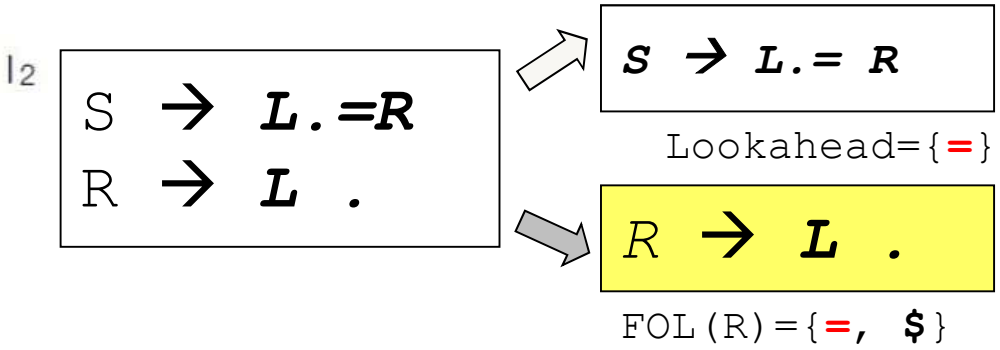
예 14: Limits of SLR(1) Parsing Power (1/3)

- **문법 :** 1. S → L = R 2. S → R 3. L → * R 4. L → id 5. R → L
- **확장 문법 : 0. S′ → S** 1. S → L = R 2. S → R 3. L → * R 4. L → id 5. R → L



FOLLOW(S) = {$}
FOLLOW(R) = {$, = }
FOLLOW(L) = {$, =}

예 14: Limits of SLR(1) Parsing Power (2/3)



shift-reduce conflict

S => L=R => L=L => L=id => id=id
S => R => L => *R = *L => *id
S => R => L => id

.....

R 다음에 =이 나타나는 *right sentential form*은 존재하지 않음.

예 14: Limits of SLR(1) Parsing Power (3/3)

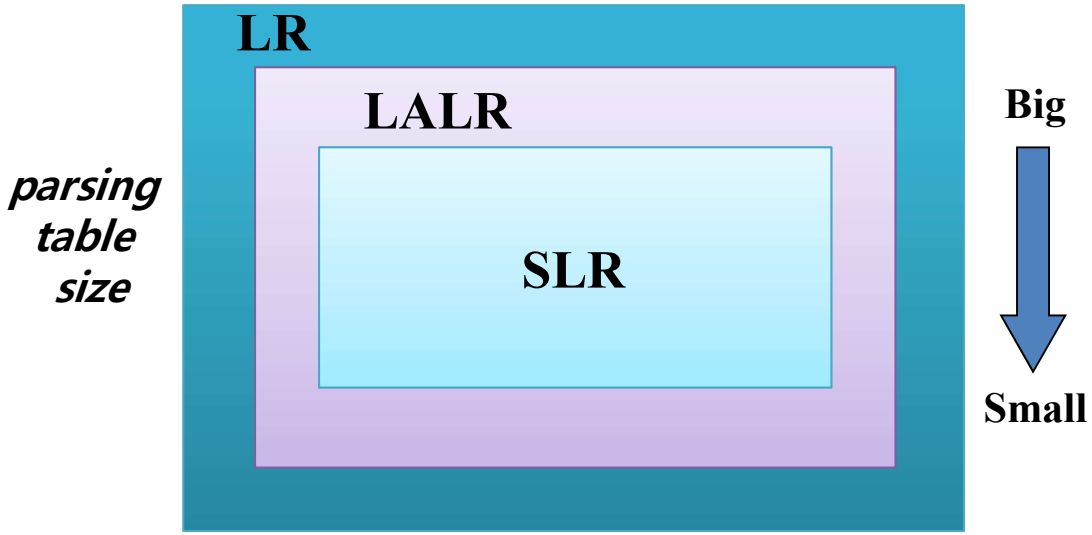
상태	구문 분석기 행동				GOTO 함수		
	=	*	id	$	S	R	L
0		s4	s5		1	3	2
1				acc			
2	r5, s6			r5			
3				r2			
4		s4	s5			7	8
5	r4			r4			
6		s4	s5			9	8
7	r3			r3			
8	r5			r5			
9				r1			

0. S' → S
1. S → L = R
2. S → R
3. L → * R
4. L → id
5. R → L

FOLLOW(S) = {$}
FOLLOW(R) = {$, = }
FOLLOW(L) = {$, =}

LR Parsing

- **SLR(*Simple* LR)**
- **LALR (*Lookahead* LR)**
  - *more general* (used in **Yacc**)
- **CLR (*Canonical* LR) 또는 general LR**



SLR(1) *versus* LR(1)

SLR(1)의 경우

- DFA를 구성하고 나서 lookahead를 적용
  - 즉, DFA를 구성할 때 lookahead를 무시

LR(1)의 경우

- DFA를 구성할 때부터 lookahead를 포함시킴
  - LR(0) item + ***lookahead*** = LR(1) item

엉뚱한 기호 말고 진짜 나타날 기호는 무엇?

- 식당에서 full-course 요리를 시켰을 때
  - 전채(appetizer) 를 먹고 나면 다음 번엔 ?
  - 후식 ?

SLR(1) Parsing 단점

SLR(1) treats all occurrences of a RHS on stack as identical. Only a few of these reductions may lead to a successful parse.

Example:

S	$\longrightarrow$	$A\mathtt{a}A\mathtt{b}$		A	$\longrightarrow \epsilon$
S	$\longrightarrow$	$B\mathtt{b}B\mathtt{a}$		B	$\longrightarrow \epsilon$

$$I_0 = \{[S' \rightarrow \bullet\, S], [S \rightarrow \bullet\, A\mathtt{a}A\mathtt{b}], [S \rightarrow \bullet\, B\mathtt{b}B\mathtt{a}], [A \rightarrow \bullet\,], [B \rightarrow \bullet\,]\}$$

Since *FOLLOW*(A) = *FOLLOW*(B), we have reduce/reduce conflict in state 0.

LR(1) Item Sets

Construct **LR(1) items** of the form [$A \rightarrow \alpha \bullet \beta$, a], which means:

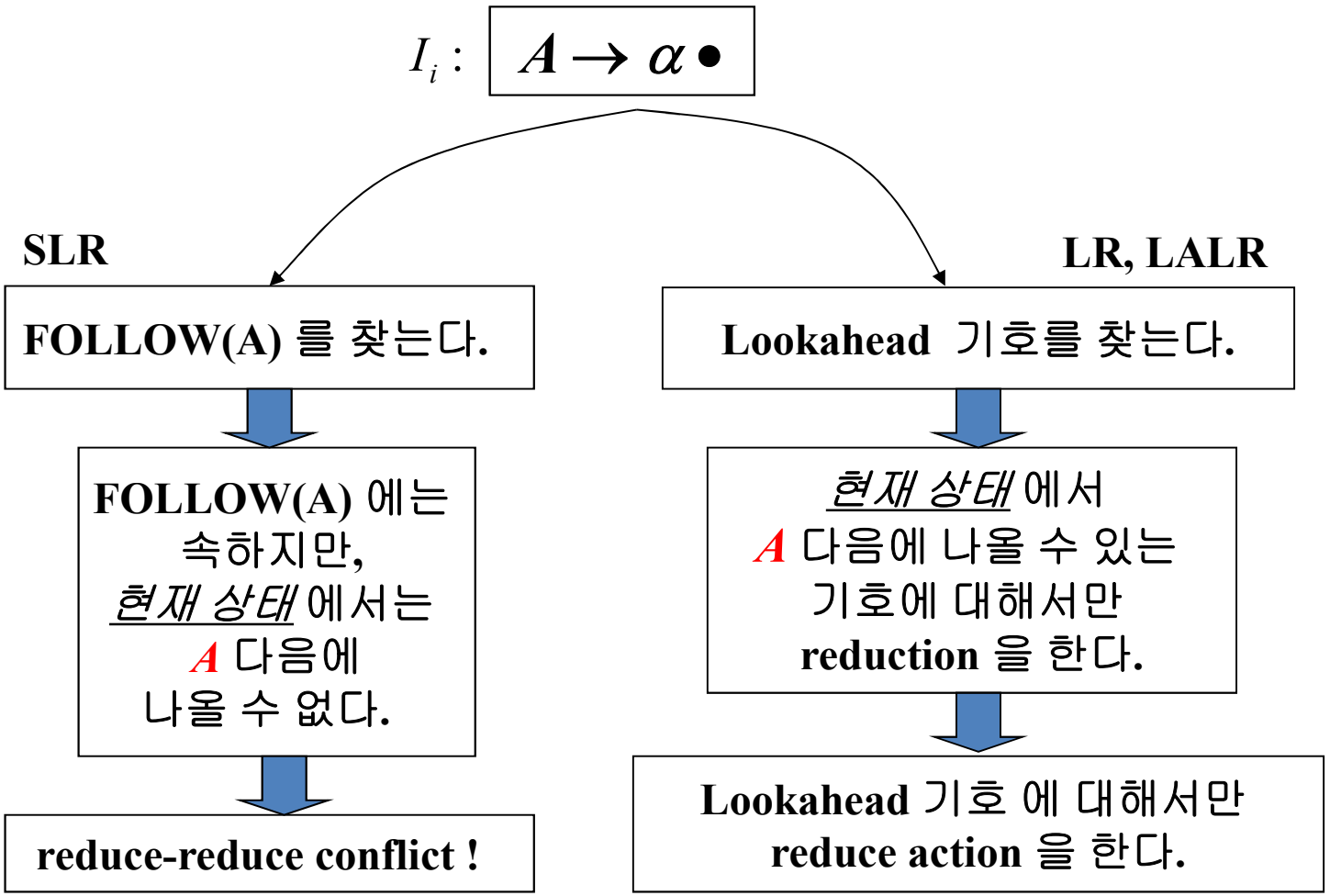
The item $A \rightarrow \alpha \bullet \beta$, can be chosen
when the next token on input stream is ***a***.

S	$\longrightarrow$	$A\mathtt{a}A\mathtt{b}$	A	$\longrightarrow$	ϵ
S	$\longrightarrow$	$B\mathtt{b}B\mathtt{a}$	B	$\longrightarrow$	ϵ

An example LR(1) item set:

$$I_0 = \{[S' \rightarrow \bullet S, \$], [S \rightarrow \bullet A\mathtt{a}A\mathtt{b}, \$], [S \rightarrow \bullet B\mathtt{b}B\mathtt{a}, \$],$$
$$[A \rightarrow \bullet, \mathtt{a}], [B \rightarrow \bullet, \mathtt{b}]\}.$$

어떤 input symbol 에 대해 reduce 할 것인가?



Finite Automata of LR(1) Items(1/2)

- **LR(1) item**
  - [$A \rightarrow \alpha \bullet \beta$, *a*] 로 표기
    - $A \rightarrow \alpha \bullet \beta$ 는 LR(0) item, $A \rightarrow \alpha\beta \in P$
    - *a* 는 lookahead (*token*), $a \in \{V_T \cup \$\}$

- **Definition of LR(1) transitions**

  Start State: [$S' \rightarrow \bullet S$, $]

Finite Automata of LR(1) Items(2/2)

Definition of LR(1) transitions

$[A \rightarrow \alpha \bullet \mathbf{X}\ \gamma,\ \boldsymbol{a}] \xrightarrow[\mathbf{X} = \textit{terminal or nonterminal}]{} [A \rightarrow \alpha\ \mathbf{X} \bullet \gamma,\ \boldsymbol{a}]$

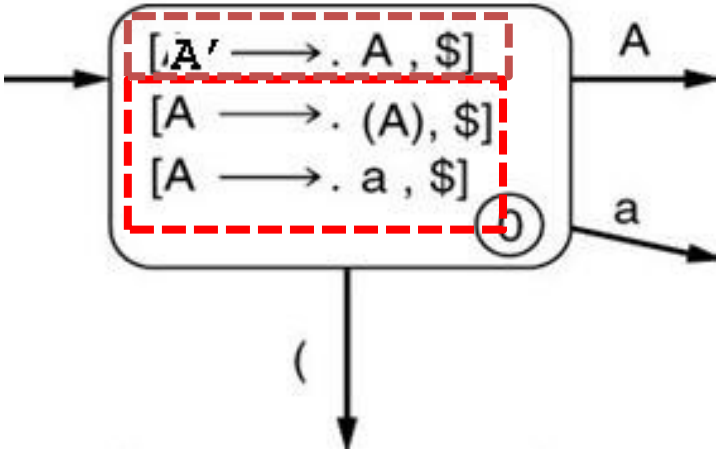
$[A \rightarrow \alpha \bullet B\gamma,\ a] \xrightarrow[(\textit{closure item}\text{을 구할 때})]{\varepsilon\textit{-transition}} [B \rightarrow \bullet\ \beta,\ \boldsymbol{b}]$

$\mathbf{B} = \textit{nonterminal}$ (***kernel item***)

for every token ***b*** in First(γa)

예 14 (1/2)

A → (A) | a



초기 상태(0)에서
kernel item [A′→.A,$]의 closure item 구하기

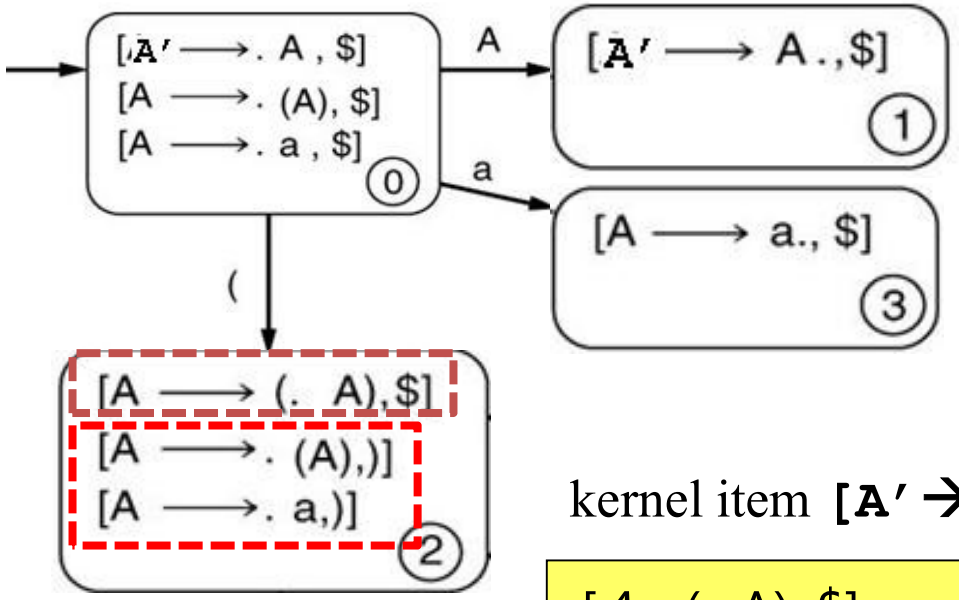
$[A' \rightarrow \bullet A, \$] \xrightarrow{\varepsilon} [A \rightarrow \bullet(A), \$]$

$B=A, \gamma=\varepsilon, a=\$$

$\text{First}(\gamma a) = \text{First}(\varepsilon\$) = \{\$\}$

$[A \rightarrow \alpha \bullet B\gamma, a] \xrightarrow{\varepsilon} [B \rightarrow \bullet \beta, \mathit{First}(\gamma a)]$

예 14 (2/2)



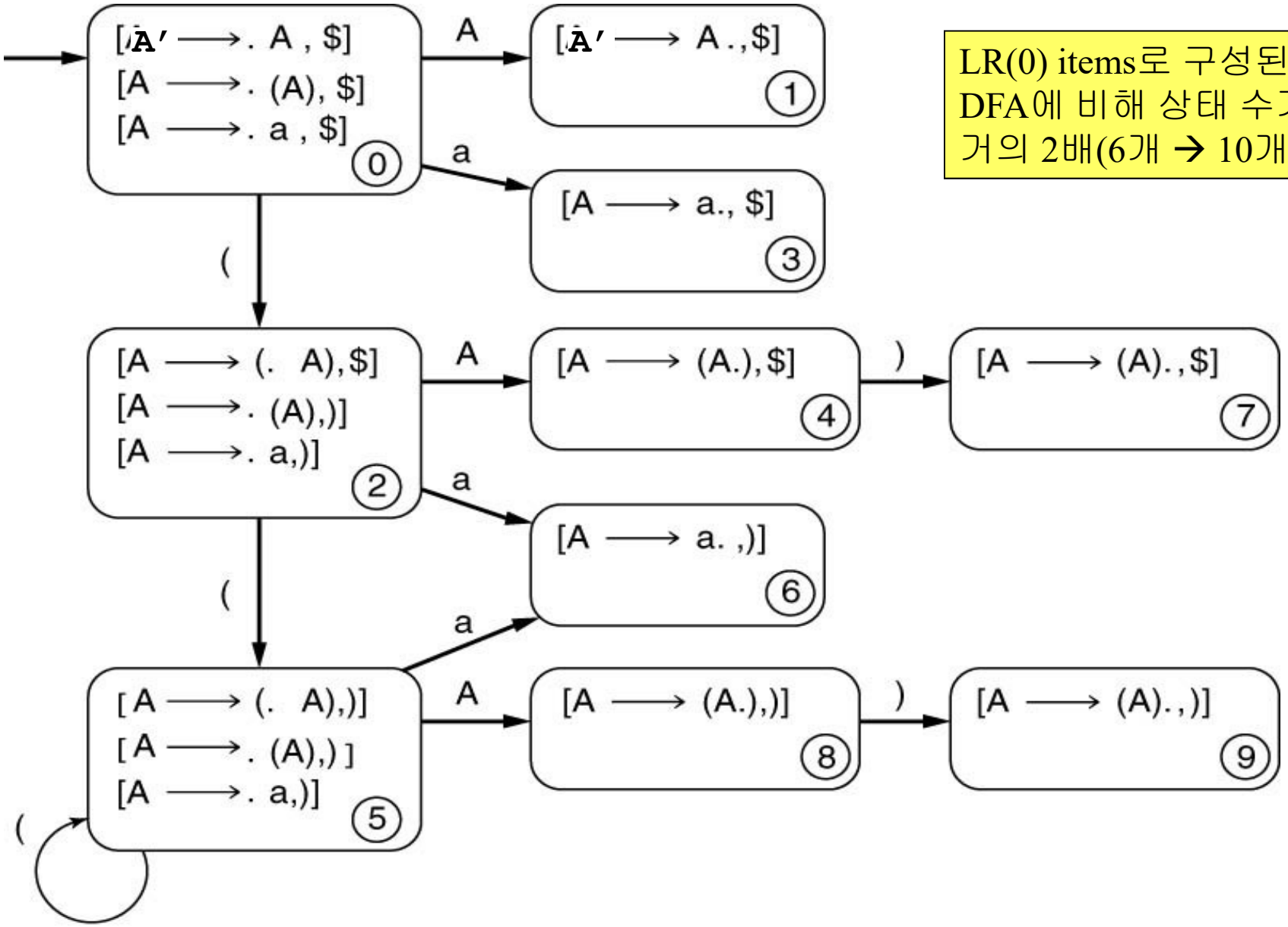
상태 2에서
kernel item **[A' → (.A) , $]** 의 closure item 구하기

$[A \rightarrow (\bullet\ A),\ \$] \xrightarrow{\varepsilon} [A \rightarrow \bullet (A),\ )]$

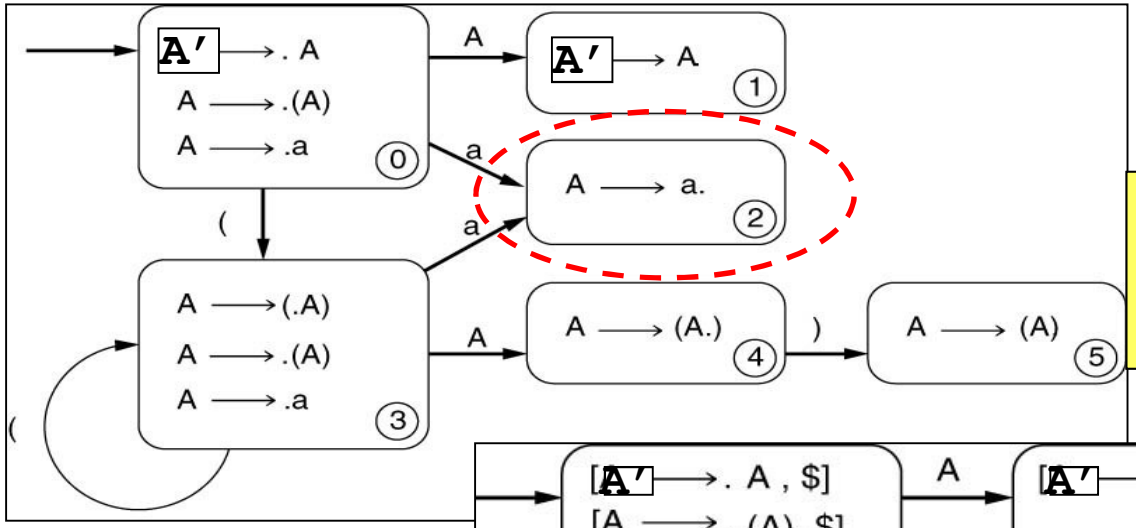
$B=A,\ \gamma=\ )\ ,\ a=\$$

$\text{First}(\gamma a) = \text{First}(\ )\$) = \{\ )\ \}$

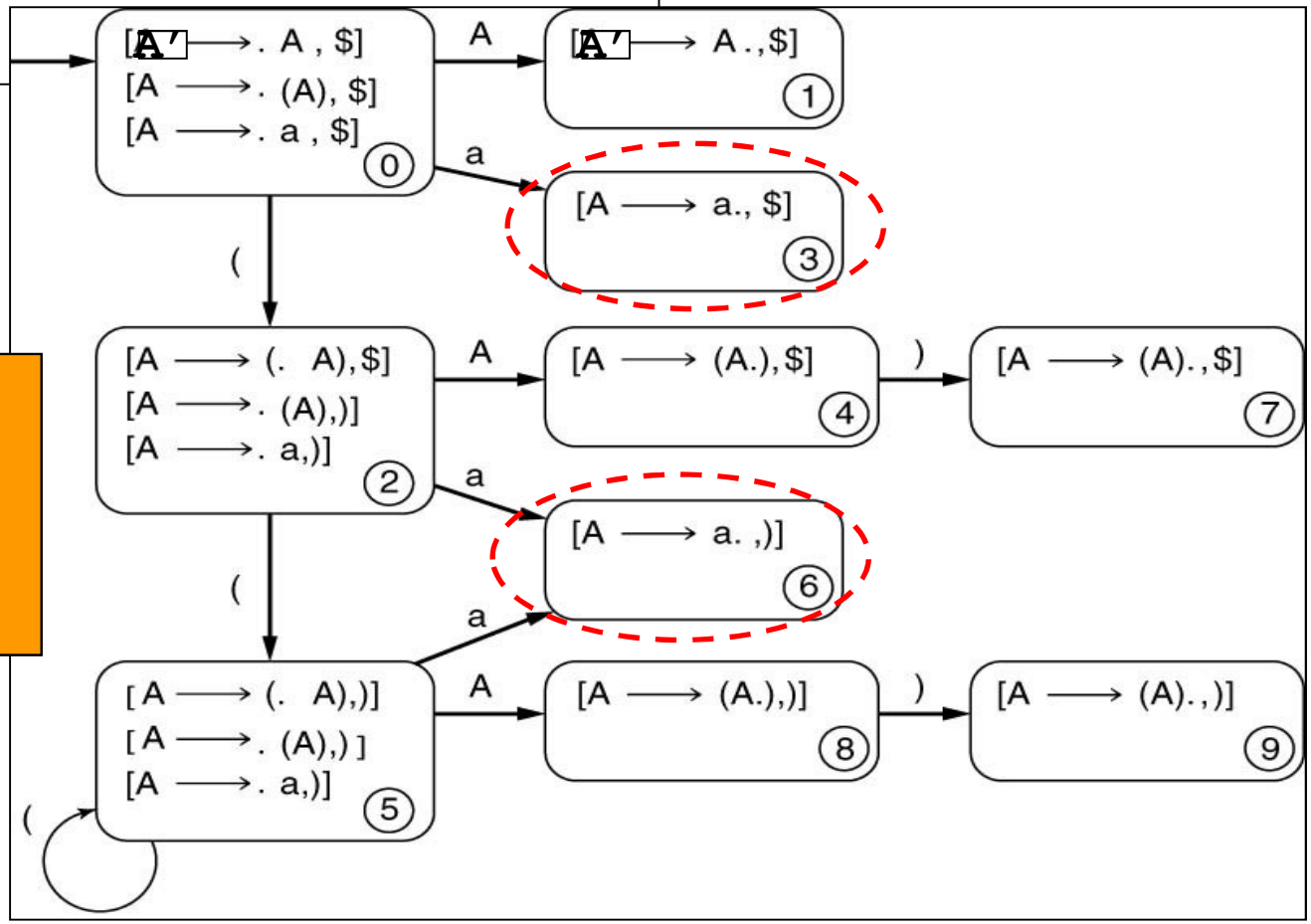
$[A \rightarrow \alpha \bullet B\gamma,\ a] \xrightarrow{\varepsilon} [B \rightarrow \bullet\ \beta,\ First\ (\gamma a)]$



LR(0) items로 구성된 DFA에 비해 상태 수가 거의 2배(6개 → 10개).



DFA of sets of LR(0) items



DFA of sets of LR(1) items

예 15: CLR(1) parsing (1/4) – Canonical Collection of LR(1) items

- **문법**

  1. S → CC 2. C → cC 3. C → d

- **확장 문법**

  0. S' → S 1. S → CC 2. C → cC 3. C → d

- Canonical Collection of LR(1) items

  I0 = CLOSURE([S′ →•S, $])

  = {[S′ →•S, $], [S →•CC, $], [C →•cC, c/d], [C →•d, c/d]}

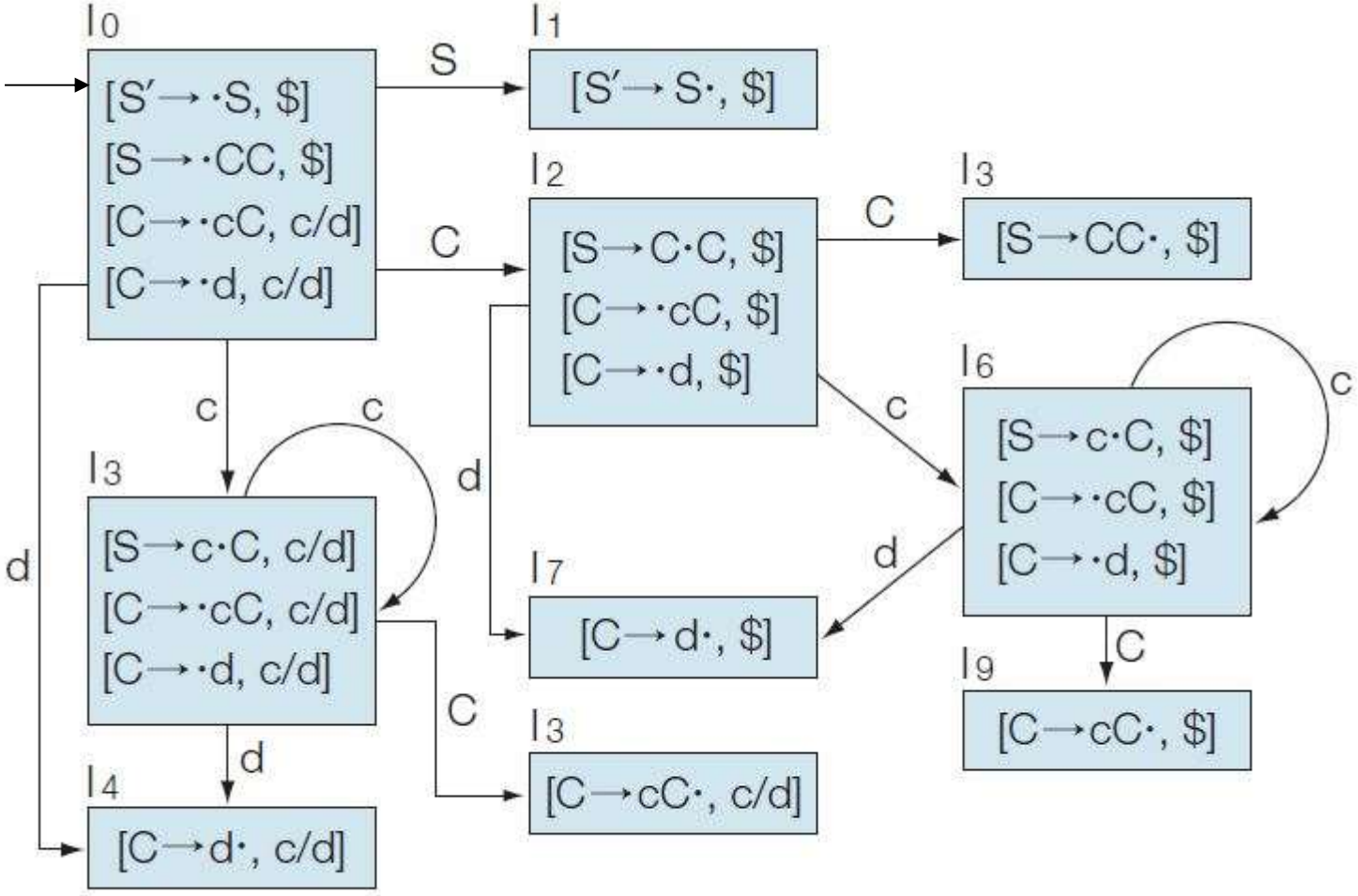
  CLOSURE(I0, S) = I1 = {[S′ →S•, $]}

  CLOSURE(I0, C) = I2 = {[S →C•C, $], [C →•cC, $], [C →•d, $]}

  CLOSURE(I0, c) = I3 = {[C →c•C, c/d], [C →•cC, c/d], [C →•d, c/d]}

  CLOSURE(I0, d) = I4 = {[C →d•, c/d]}

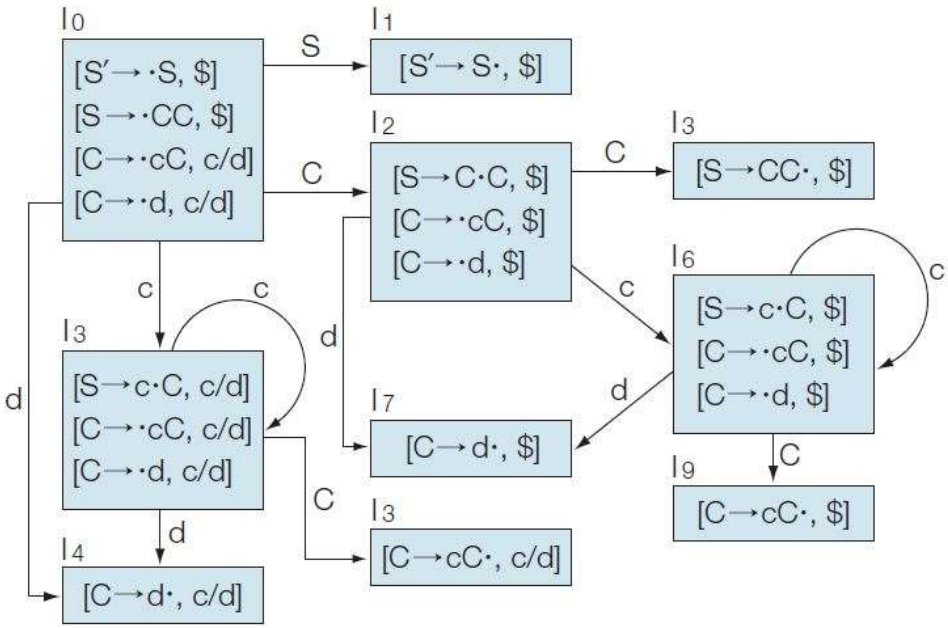
예 15: CLR(1) parsing (2/4) - DFA of LR(1) items



예 15: CLR(1) parsing (3/4) – parsing table

상태	구문 분석기 행동			GOTO 함수	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

0. S' → S
1. S → CC
2. C → cC
3. C → d



예 15: CLR(1) parsing (4/4) – syntax analysis

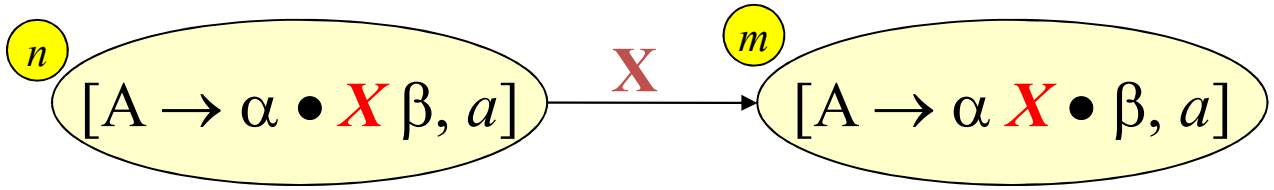
단계	스택	입력 기호	구문 분석 내용
0	0	ccdd$	이동 3
1	0c3	cdd$	이동 3
2	0c3c3	dd$	이동 4
3	0c3c3d4	d$	감축 3
4	0c3c3C	d$	GOTO 8
5	0c3c3C8	d$	감축 2
6	0c3C	d$	GOTO 8
7	0c3C8	d$	감축 2
8	0C	d$	GOTO 2
9	0C2	d$	이동 7
10	0C2d7	$	감축 3
11	0C2C	$	GOTO 5
12	0C2C5	$	감축 1
13	0S	$	GOTO 1
14	0S1	$	수락

상태	구문 분석기 행동			GOTO 함수	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

General LR(1) Parsing Algorithm (1/3)

■ **현재 상태 *n* 이**

1. **[*A* → α • Xβ , *a*] 형태의 item을 갖고 있는 경우**
   - **X**가 *terminal*이고, 입력에서 읽어 올 *next token* 도 **X** 이면,
     - ***shift* action** : *input lookahead* 를 stack에 push
   - 다음 상태 ***m***은 [*A* → αX • β, a] 의 item을 포함

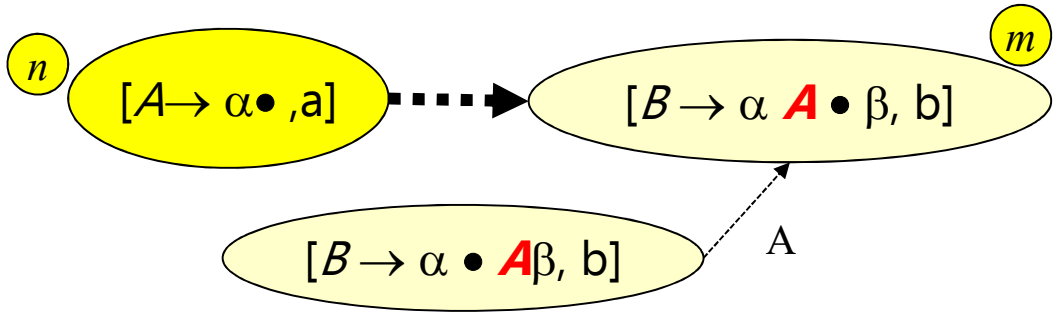


General LR(1) Parsing Algorithm (2/3)

- **현재 상태 *n* 이**

2. **complete item [$A \rightarrow \alpha \bullet, a$] 을 가지며, 입력에서 읽어 온 token이 *a* 이면,**

- ***reduce* action**: reduction by $A \rightarrow \alpha \bullet$
  - Stack에서 기호(α) 와 상태 번호를 함께 제거(*pop*)
    - nonterminal *A*를 stack에 push
- $B \rightarrow \alpha A \bullet \beta$ ($B \rightarrow \alpha \bullet A\beta$ 에서 reduction직후의 item) 를 포함하는
  - 상태 번호 ***m*** 을 push



General LR(1) Parsing Algorithm (3/3)

■ 현재 상태 *n* 이

2. complete item [$S' \rightarrow S \bullet$, \$] 을 가지며,

   입력에서 읽어 온 기호가 \$ 이면 accept!

LR(1) Grammar

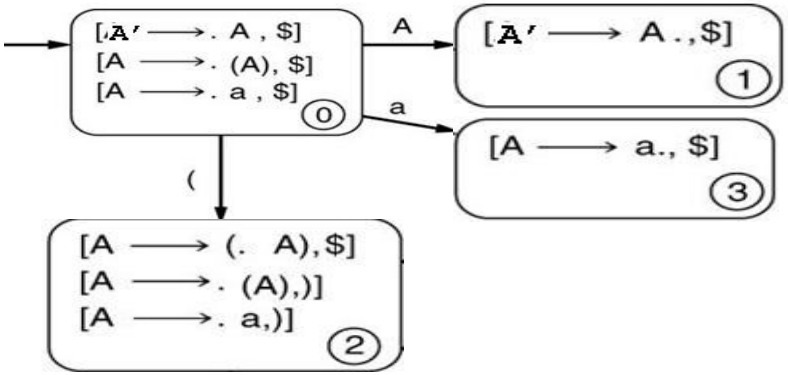
- **A grammar is LR(1) *if and only if* the following two conditions are satisfied:**

for any state `s`,

1. **no *shift-reduce conflict***
   - For any item $[A \rightarrow \alpha \bullet \mathbf{X}\beta, a]$ in **`s`** with *X a terminal*,
     - there is no complete item $[B \rightarrow \gamma \bullet , \mathbf{X}]$ in **`s`**.

2. **no *reduce-reduce conflict***
   - There are *no* two items in **`s`** of the form
     - $[A \rightarrow \alpha \bullet, \mathbf{a}]$ and $[B \rightarrow \beta \bullet, \mathbf{a}]$.

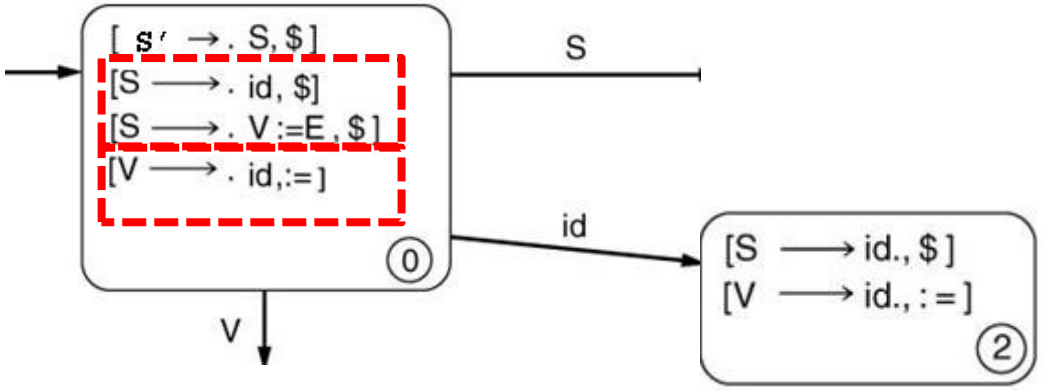
예 16: General LR(1) Parsing Table



State	Input				Goto
	(	a	)	$	A
0	s2	s3			1
1				accept	
2	s5	s6			4
3				r2	

(1) A → (A) (2) A → a

예 17 (1/2)




```
S → id | V := E
V → id
E → V | n
```

초기 상태(0)에서
kernel item [S′ → • S, $]의 Closure item 구하기

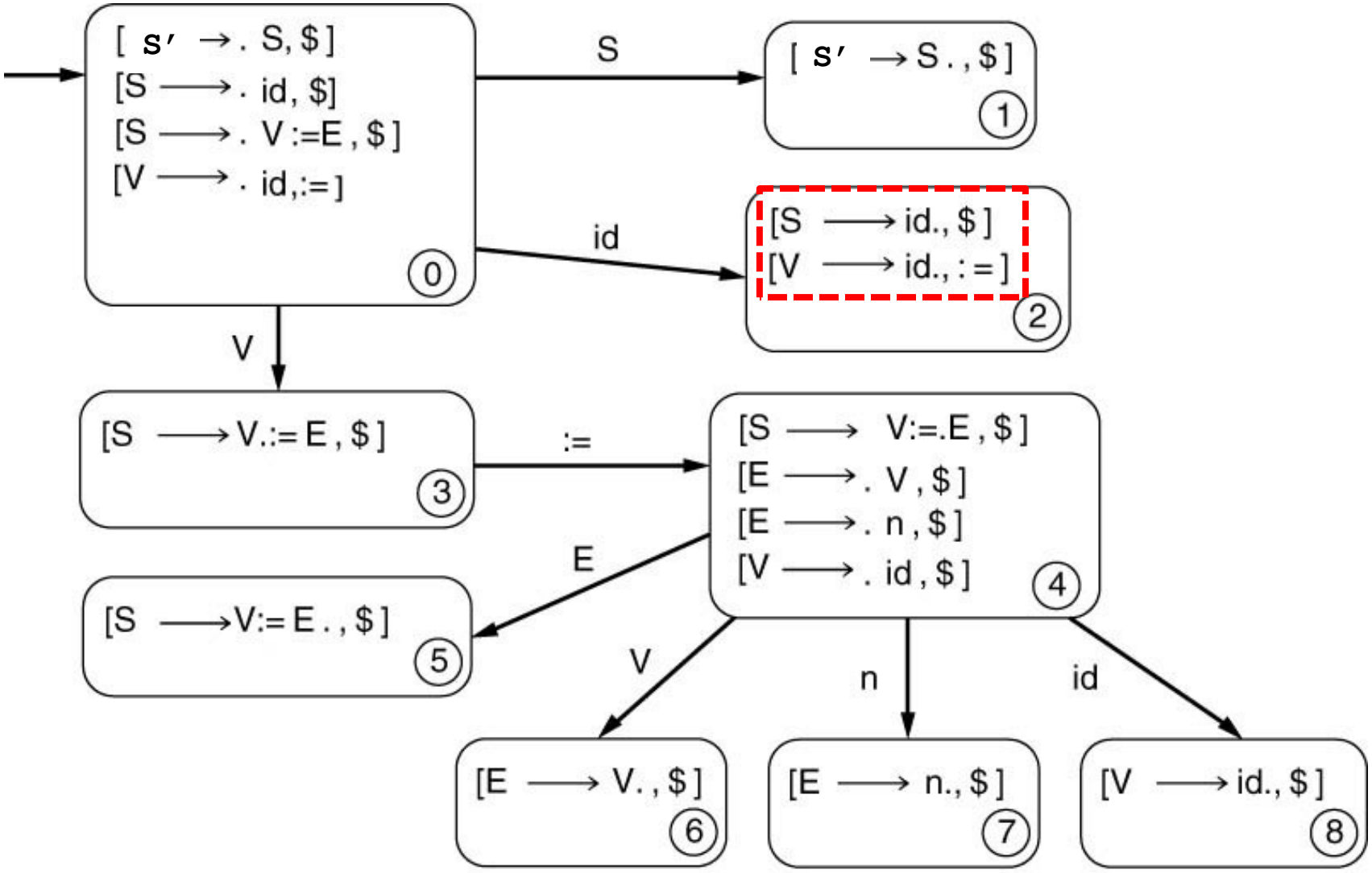
$[S \rightarrow \bullet\ V{:=}E,\ \$] \xrightarrow{\varepsilon} [V= \bullet\ id,\ :=]$

$\alpha = \varepsilon,\ B=V,\ \gamma= :=$

$\text{First}(\gamma a) = \text{First}(:= \$) = \{:=\}$

$[A \rightarrow \alpha \bullet B\gamma,\ a] \xrightarrow{\varepsilon} [B \rightarrow \bullet\ \beta,\ First(\gamma a)]$

예 17 (2/2)


[s′ → . S, $]
[S ⟶ . id, $]
[S ⟶ . V :=E , $]
[V ⟶ . id,:=]
0
S
[s′ → S . , $]
1
id
[S ⟶ id., $]
[V ⟶ id., : =]
2
V
[S ⟶ V.:= E , $]
3
:=
[S ⟶ V:=.E , $]
[E ⟶ . V , $]
[E ⟶ . n , $]
[V ⟶ . id , $]
4
E
[S ⟶ V:= E . , $]
5
V
n
id
[E ⟶ V. , $]
6
[E ⟶ n., $]
7
[V ⟶ id., $]
8

예 18: CLR(1) parsing (1/4)

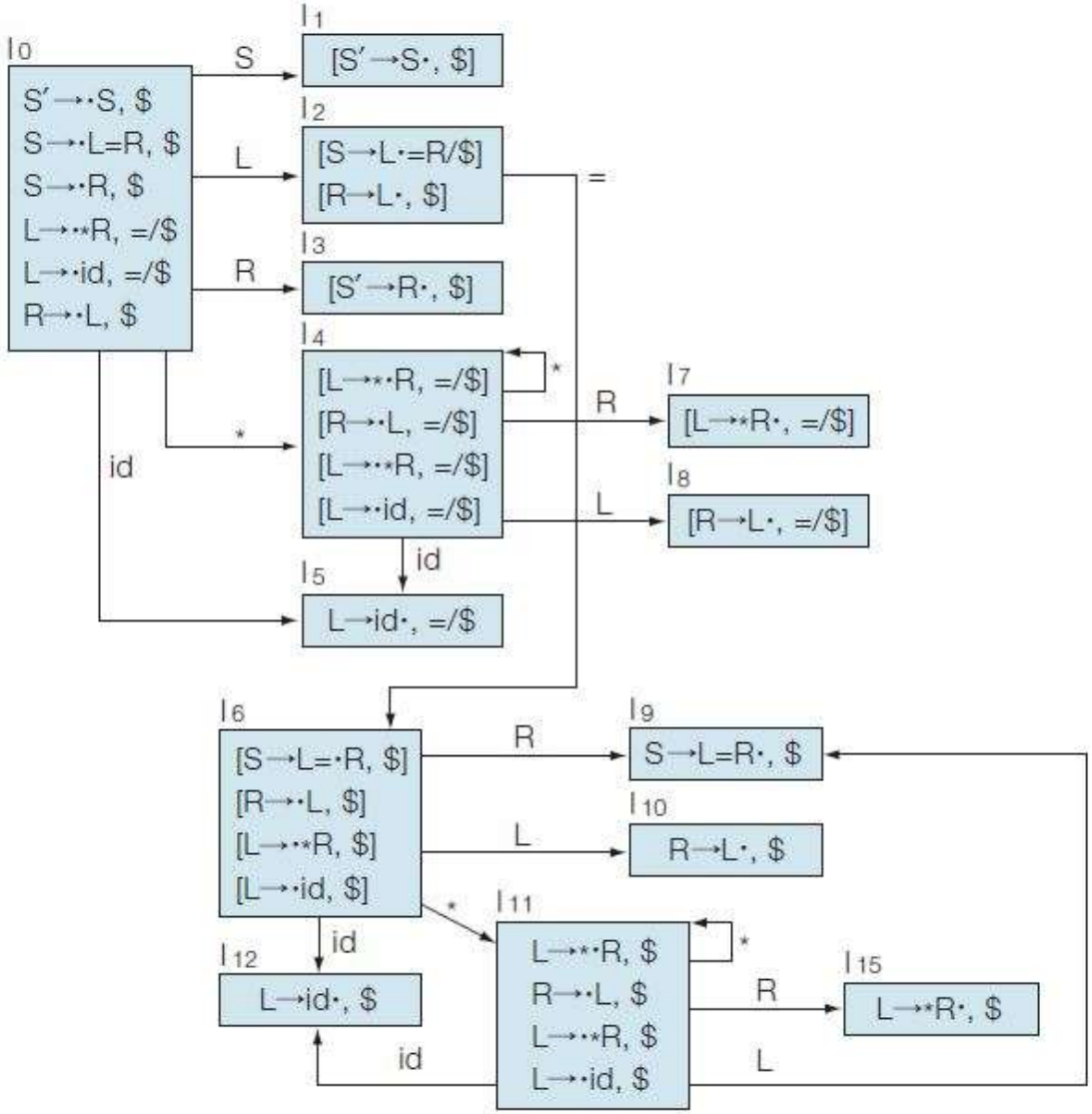
- **문법**

  1. S → L = R 2. S → R 3. L → * R 4. L → id 5. R → L

- **Augmented grammar**

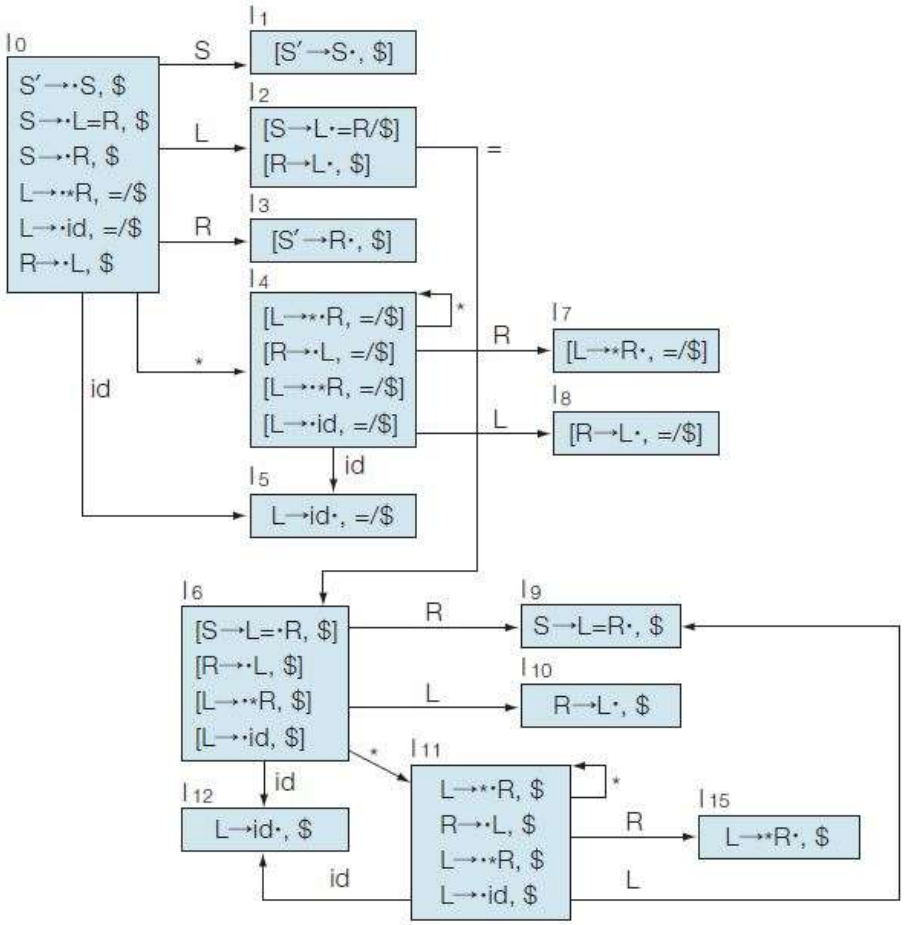
  0. S′ → S 1. S → L = R 2. S → R 3. L → * R 4. L → id 5. R → L

예 18: CLR(1) parsing (2/4) – DFA of LR(1) items



예 18: CLR(1) parsing (3/4) – parsing table

상태	구문 분석기 행동				GOTO 함수		
	=	*	id	$	S	R	L
0		s4	s5		1	3	2
1				acc			
2	s6			r5			
3				r2			
4		s4	s5			7	8
5	r4						
6		s11	s12			9	10
7	r3			r3			
8	r5			r3			
9				r1			
10				r5			
11		s11	s12			13	10
12				r4			
13				r3			

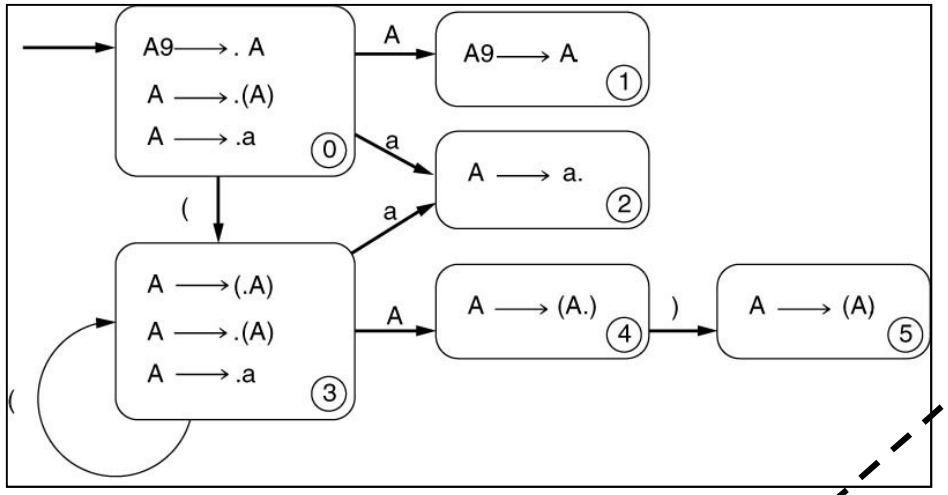


예 18: CLR(1) parsing (4/4) – syntax analysis

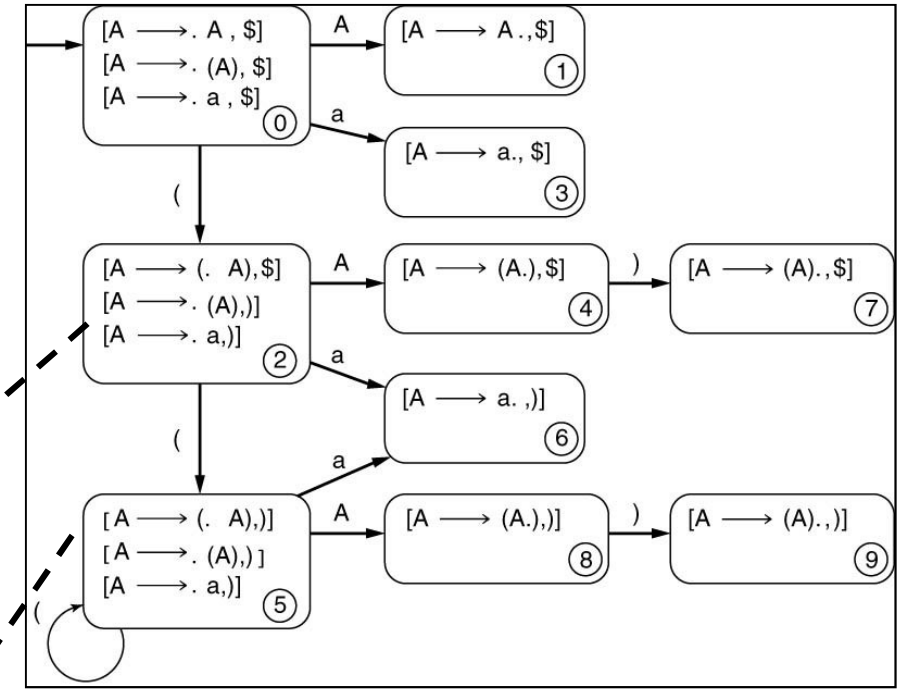
단계	스택	입력 기호	구문 분석 내용
0	0	*id=id$	이동 4
1	0*4	id=id$	이동 5
2	0*4id5	=id$	감축 4
3	0*4L	=id$	GOTO 8
4	0*4L8	=id$	감축 5
5	0*4R	=id$	GOTO 7
6	0*4R7	=id$	감축 3
7	0L	=id$	GOTO 2
8	0L2	=id$	이동 6
9	0L2=6	id$	이동 12
10	0L2=6id12	$	감축 4
11	0L2=6L	$	GOTO 10
12	0L2=6L10	$	감축 5
13	0L2=6R	$	GOTO 9
14	0L2=6R9	$	감축 1
15	0S	$	GOTO 1
16	0S1	$	수락

상태	구문 분석기 행동				GOTO 함수		
	=	*	id	$	S	R	L
0		s4	s5		1	3	2
1				acc			
2	s6			r5			
3				r2			
4		s4	s5			7	8
5	r4						
6		s11	s12			9	10
7	r3			r3			
8	r5			r3			
9				r1			
10				r5			
11		s11	s12			13	10
12				r4			
13				r3			

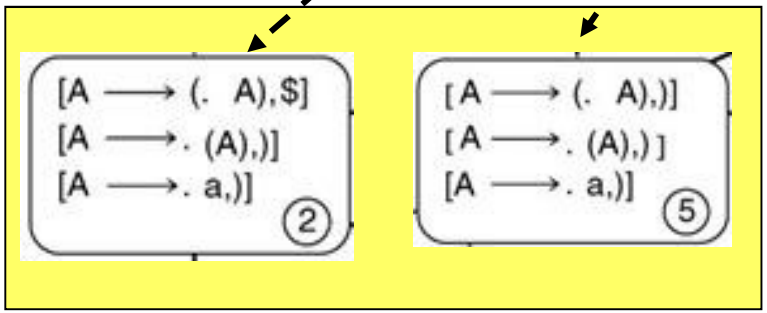
What's the problem?



DFA of LR(0) items



DFA of LR(1) items



What's the difference?

LALR(1) Parsing

LR(1) has so many states!

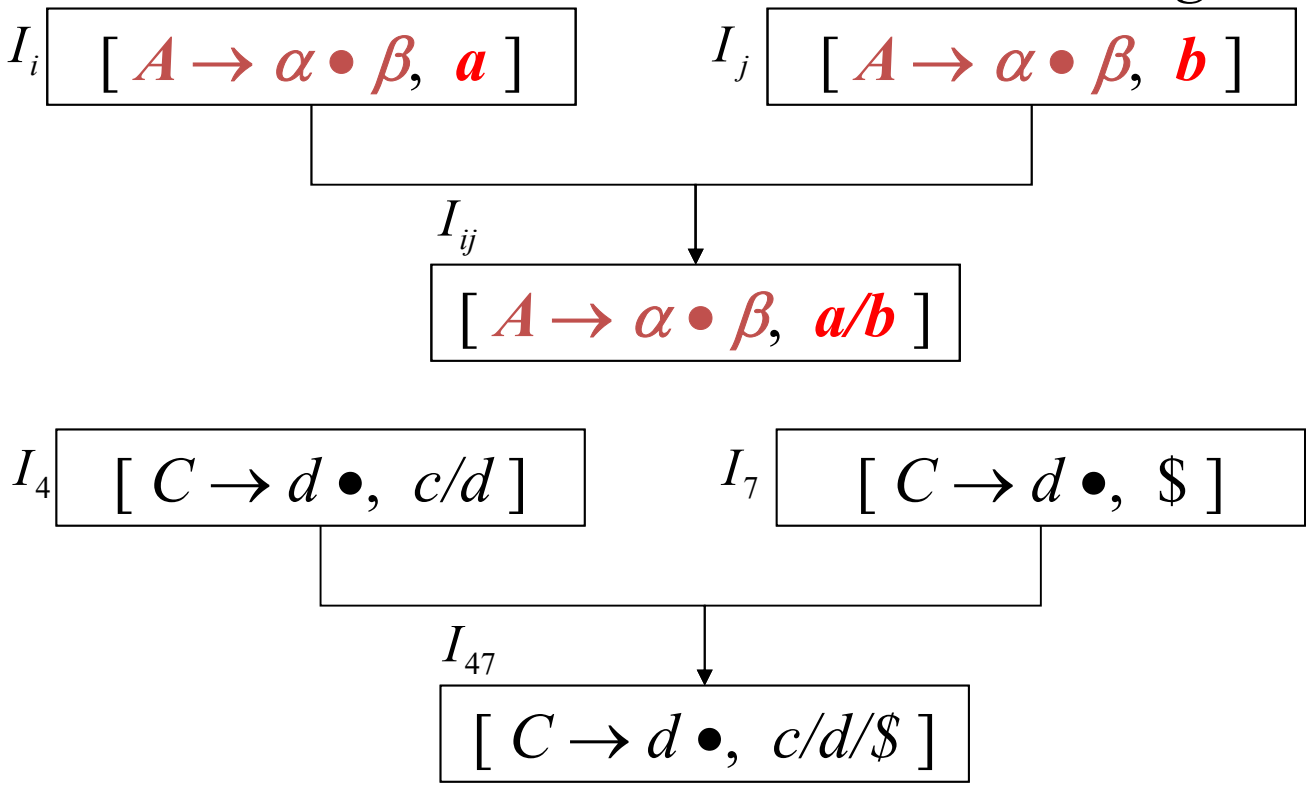
- LR(0) item은 같지만, *lookahead* 만 다른 상태들이 많음
  - 이 상태들을 모두 합치면 LR(0) item으로 이루어진 DFA로 축약됨.
- LR(0) item으로 이루어진 DFA와 LR(1) item 으로 이루어진 DFA의 차이점은
  - lookahead 포함 여부

상태 수를 줄이기 위해

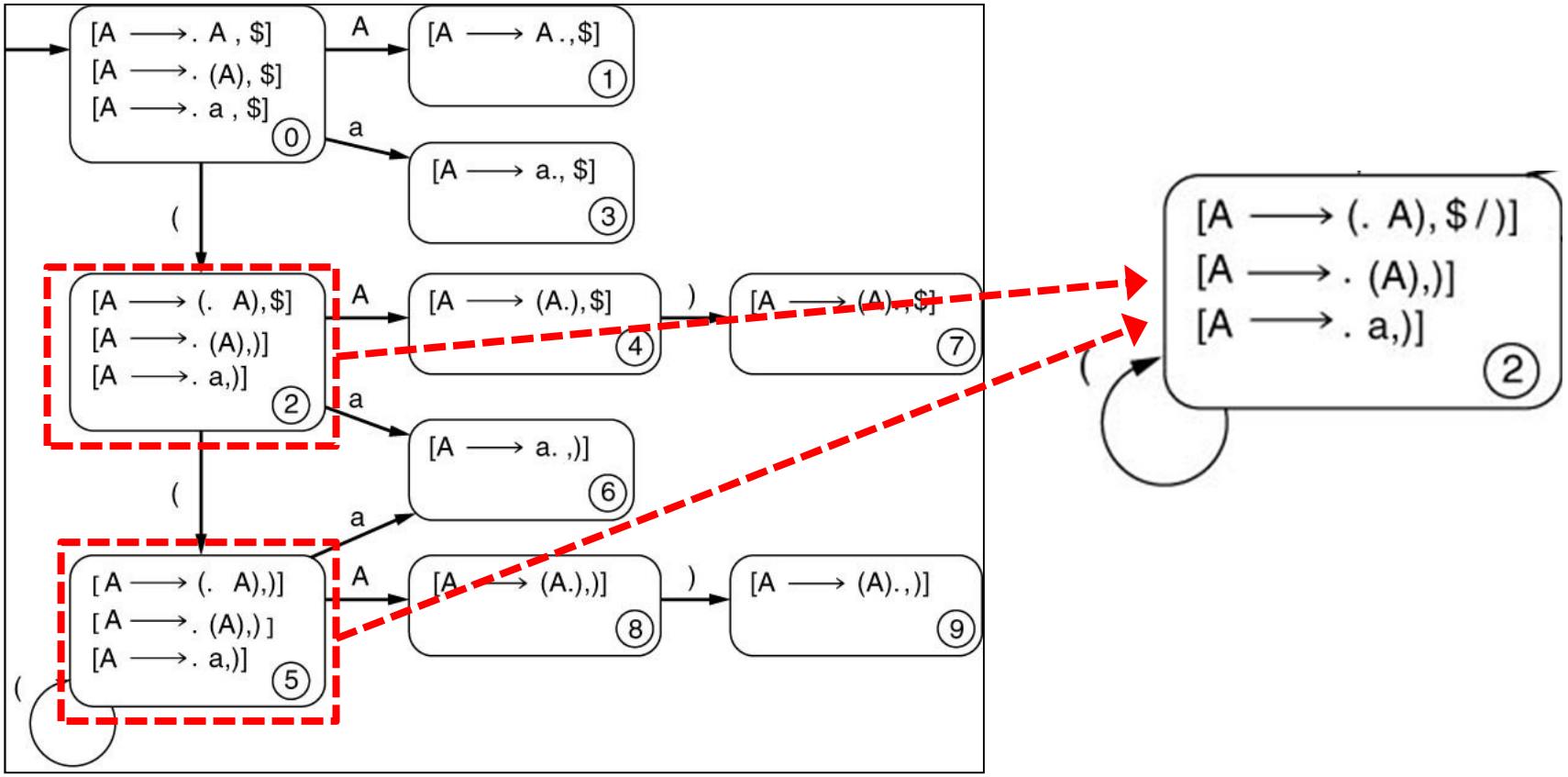
- core item이 같은 상태들을 merge 시킴
  - lookahead 가 달라 따로 취급했던 상태들을 하나의 상태로 합침
- **DFA of LALR(1) items**

LALR(1) Parsing

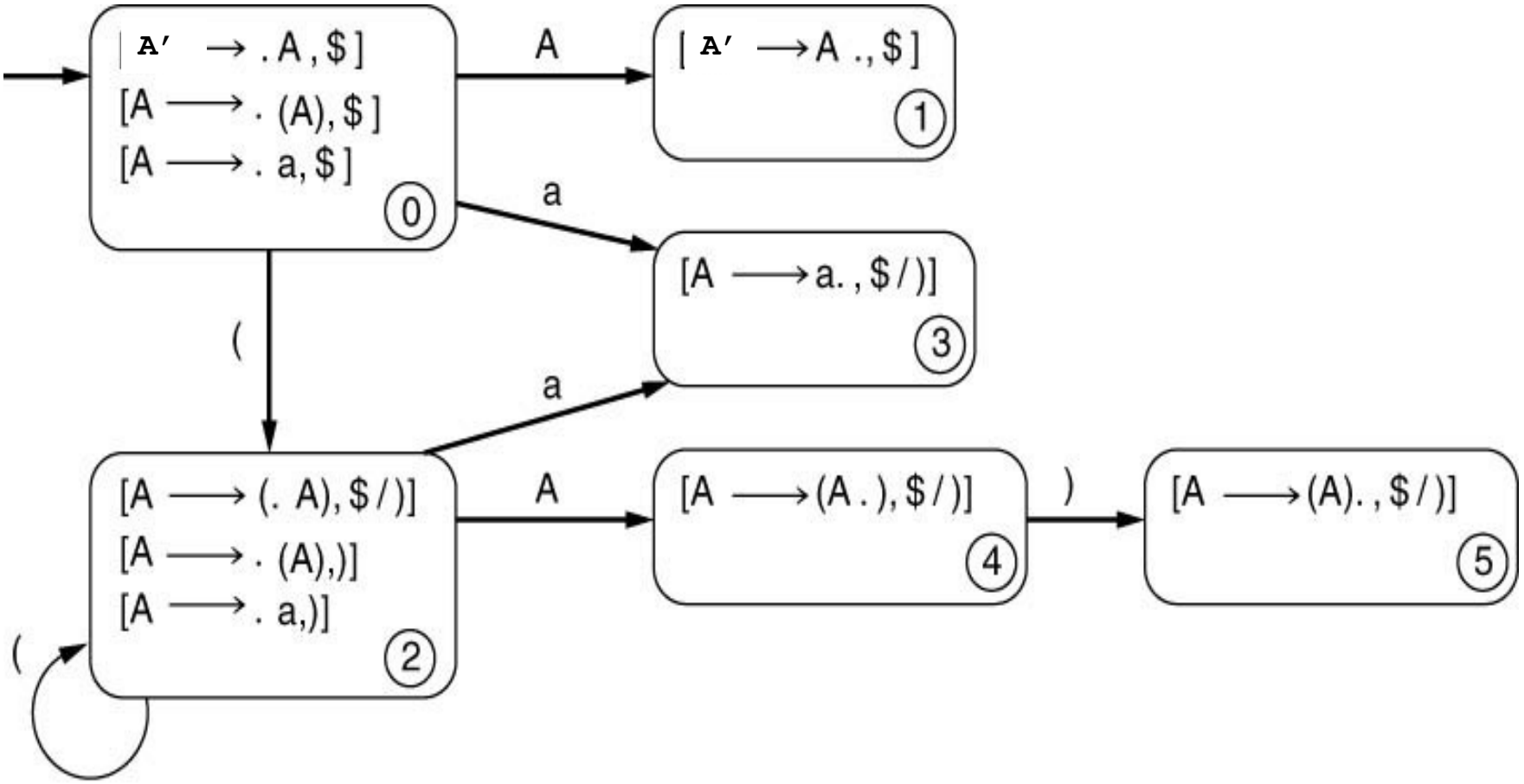
같은 *core* 를 갖고 있는 서로 다른 상태를 *merge* 시킴.



Identifying states 2 and 4 gives the state of LALR(1) items



예 19: DFA of LALR(1) items



예 20: CLR table을 LALR table로 바꾸기 (1/2)

표 6-5 CLR 파싱표

상태	구문 분석기 행동			GOTO 함수	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

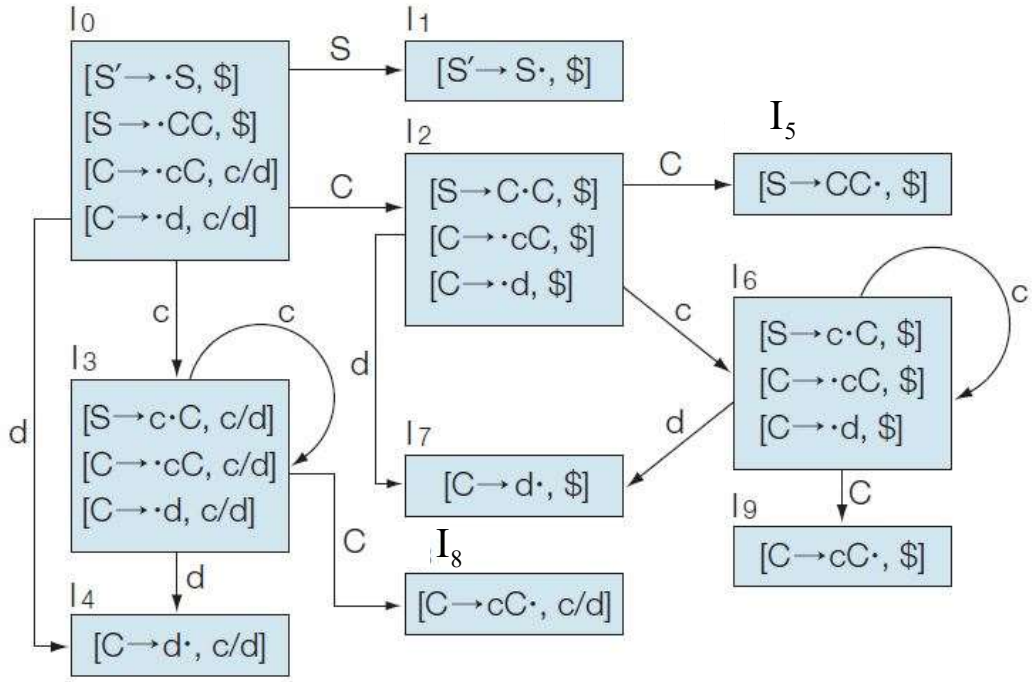


그림 6-7 CLR 정규 항목 집합과 GOTO 그래프

I3 ∪ I6 ⇒ I36 = {[C → c•C, c/d/$], [C →•cC, c/d/$], [C →•d, c/d/$]}

I4 ∪ I7 ⇒ I47 = [[C → d•, c/d/$]}

I8 ∪ I9 ⇒ I89 = {[C → cC•, c/d/$]}

예 20: CLR table을 LALR table로 바꾸기 (2/2)

표 6-5 CLR 파싱표

상태	구문 분석기 행동			GOTO 함수	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

표 6-8 LALR 파싱표

상태	구문 분석기 행동			GOTO 함수	
	c	d	$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

LR(1) and LALR(1) Parsing

LR(1) parsing: Parse tables built using LR(1) item sets.

LALR(1) parsing: *Look Ahead* LR(1)

Merge LR(1) item sets; the build parsing table.

Typically, LALR(1) parsing tables are much smaller than LR(1) parsing table.

$SLR(1) \subset LALR(1) \subset LR(1)$.

$LL(1) \not\subseteq SLR(1)$, but $LL(1) \subset LR(1)$.

Construct the DFA of LR(0) items for the following grammar

$$P \rightarrow b\ D\ ;\ S\ e$$
$$D \rightarrow d\ ;\ D \mid d$$
$$S \rightarrow s\ ;\ S \mid s$$