

# Compilers

## **CS414-2010-05**

### ***LR Parsing***

David Galles

Department of Computer Science  
University of San Francisco

# 05-0: Parsing

---

- LL(1) – Left-to-right, Leftmost derivation, 1-symbol lookahead parsers
  - Need to guess which rule to apply after looking at only the first element in the rule
- LR parsers – Left-to-right, Rightmost derivation parsers
  - Look at the entire right-hand side of the rule before deciding which rule to apply

# 05-1: LR Parsing

---

- Maintain a stack
- Shift terminals from the input stream to the stack, until the top of the stack is the same as the right-hand side of a rule
- When the top of the stack is the same as the right-hand side of a rule *reduce* by that rule – replace the right-hand side of the rule on the stack with the left-hand side of the rule.
- Continue shifting elements and reducing by rules, until the input has been consumed and the stack contains only the initial symbol

## 05-2: LR Parsing Example

---

$$(0) E' \rightarrow E\$$$

$$(1) E \rightarrow E + T$$

$$(2) E \rightarrow T$$

$$(3) T \rightarrow T * \text{num}$$

$$(4) T \rightarrow \text{num}$$

$$3 + 4 * 5\$$$

$$3 * 4 + 5\$$$

## 05-3: LR Parsing

---

- How do we know when to shift, and when to reduce?
- Use a Deterministic Finite Automaton
  - Combination of DFA and a stack is called a Push-down automaton
- We will put both states and symbols on the stack
- When the end-of-file marker is shifted, accept the string

# 05-4: LR Parsing Example

(0)  $E' \rightarrow E\$$

(1)  $E \rightarrow E + T$

(2)  $E \rightarrow T$

(3)  $T \rightarrow T * \text{num}$

(4)  $T \rightarrow \text{num}$

	num	+	*	\$	$E$	$T$
1	s2				g3	g4
2		r(4)	r(4)	r(4)		
3		s7		a		
4		r(2)	s5	r(2)		
5	s6					
6		r(3)	r(3)	r(3)		
7	s2					g8
8		r(1)	s(5)	r(1)		

## 05-5: LR Parsing

---

- LR(0) Parsers. Reduce as soon as the top of the stack is the same as the left-hand side of a rule
- SLR(1) Parsers. More powerful than LR(0) – adds some lookahead information
- LR(1) Parsers. More powerful than SLR(1) – adds more sophisticated lookahead information
- LALR Parsers. *Almost* as powerful as LR(1), but uses much less memory (smaller table sizes)

## 05-6: LR(0) Parsing

---

- Reads the input file Left-to-Right  $\underline{L}R(0)$
- Creates a Rightmost derivation  $L\underline{R}(0)$
- No Lookahead (0-symbol lookahead)  $LR(\underline{0})$

LR(0) parsers are the simplest of the LR parsers

# 05-7: LR Parsing Example

---

(0)  $S' \rightarrow S\$$

(1)  $S \rightarrow AA$

(2)  $S \rightarrow bc$

(3)  $A \rightarrow baA$

(4)  $A \rightarrow c$

# 05-8: LR Parsing Example

---

$$S' \rightarrow S\$$$

$$S \rightarrow AA$$

$$S \rightarrow bc$$

$$A \rightarrow baA$$

$$A \rightarrow c$$

	a	b	c
$S'$		$S' \rightarrow S\$$	$S' \rightarrow S\$$
$S$		$S \rightarrow bc$ $S \rightarrow AA$	$S \rightarrow AA$
$A$		$A \rightarrow baA$	$A \rightarrow c$

Not LL(1)!

## 05-9: LR(0) Items

---

- An LR(0) item consists of
  - A rule from the CFG
  - A “.” in the rule, which indicates where we currently are in the rule
- $S \rightarrow ab . c$ 
  - Trying to parse the rule  $S \rightarrow abc$
  - Already seen “ab”, looking for a “c”

# 05-10: LR(0) States & Transitions

---

(0)  $S' \rightarrow S\$$

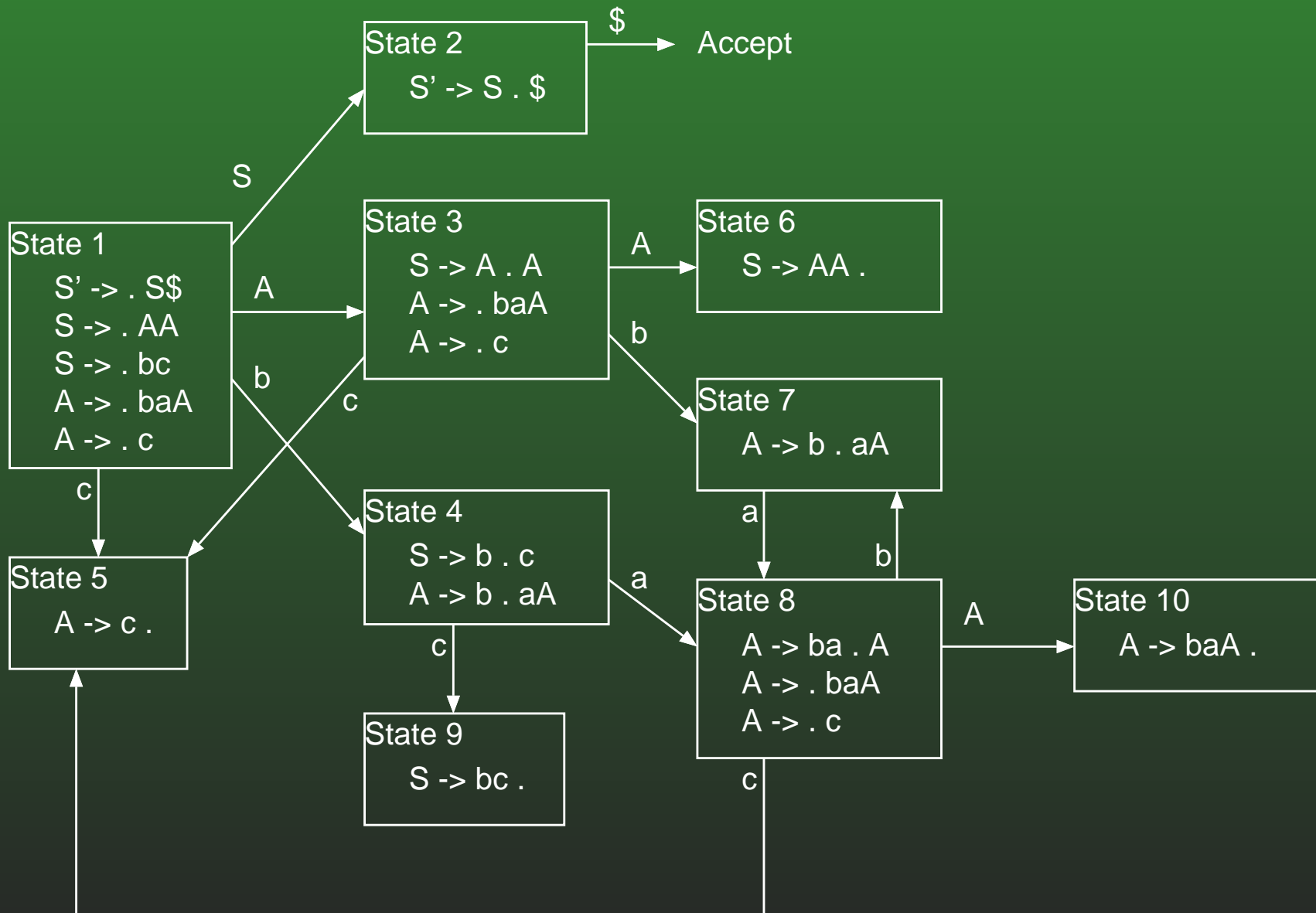
(1)  $S \rightarrow AA$

(2)  $S \rightarrow bc$

(3)  $A \rightarrow baA$

(4)  $A \rightarrow c$

# 05-11: LR(0) States & Transitions



# 05-12: LR(0) Parse Table

---

	a	b	c	\$	$S$	$A$
1		s4	s5		g2	g3
2				accept		
3		s7	s5			g6
4	s8		s9			
5	r(4)	r(4)	r(4)	r(4)		
6	r(1)	r(1)	r(1)	r(1)		
7	s8					
8		s7	s5			g10
9	r(2)	r(2)	r(2)	r(2)		
10	r(3)	r(3)	r(3)	r(3)		

## 05-13: Closure & Transitions

---

- Two basic operations for creating LR(0) parsers:
  - Finding the *closure* of a state
  - Finding the transitions out of a state

## 05-14: Closure

---

1. For each item in the state of the form  $S \rightarrow \alpha \cdot S_1\beta$ , where  $\alpha$  and  $\beta$  are (possibly empty) strings of terminals and non-terminals, and  $S_1$  is a non-terminal:
  - For each rule of the form  $S_1 \rightarrow \gamma$  add the item  $S_1 \rightarrow \cdot \gamma$  if it is not already there
2. If any items were added in step 1, go back to step 1 and repeat

## 05-15: Closure

---

- If a “.” appears right before the non-terminal  $S$  in an item
  - Add items for all  $S$  rules to the state, with the “.” at the beginning of the rule
- Repeat until no more items can be added

## 05-16: Finding Transitions

---

1. If the end-of-file terminal \$ appears before the "." in some item in the original state, create a transition from the original state to an "accept" state, transitioning on \$.
2. For each terminal a (other than \$) that appears before the "." in some item in the original state:
  - Create a new empty state.
  - For each item in the original state of the form  $S \rightarrow \alpha . a \gamma$ , where  $\alpha$  and  $\gamma$  are (possibly empty) strings of terminals and non-terminals, add the item  $S \rightarrow \alpha a . \gamma$  to the new state.
  - Find the closure of the new state.
  - Add a transition from the original state to the new state, labeled with a.

## 05-17: Finding Transitions

---

- If a “.” appears just before a terminal  $a$  in at least one item:
  - Create a new state
  - Add all items where the “.” is just before an  $a$
  - Move the “.” past the  $a$  in the new state
  - Find the closure of the new state
  - Add a transition to the new state, labeled with an  $a$ .

## 05-18: Another LR(0) Example

---

$$(0) E' \rightarrow E\$$$

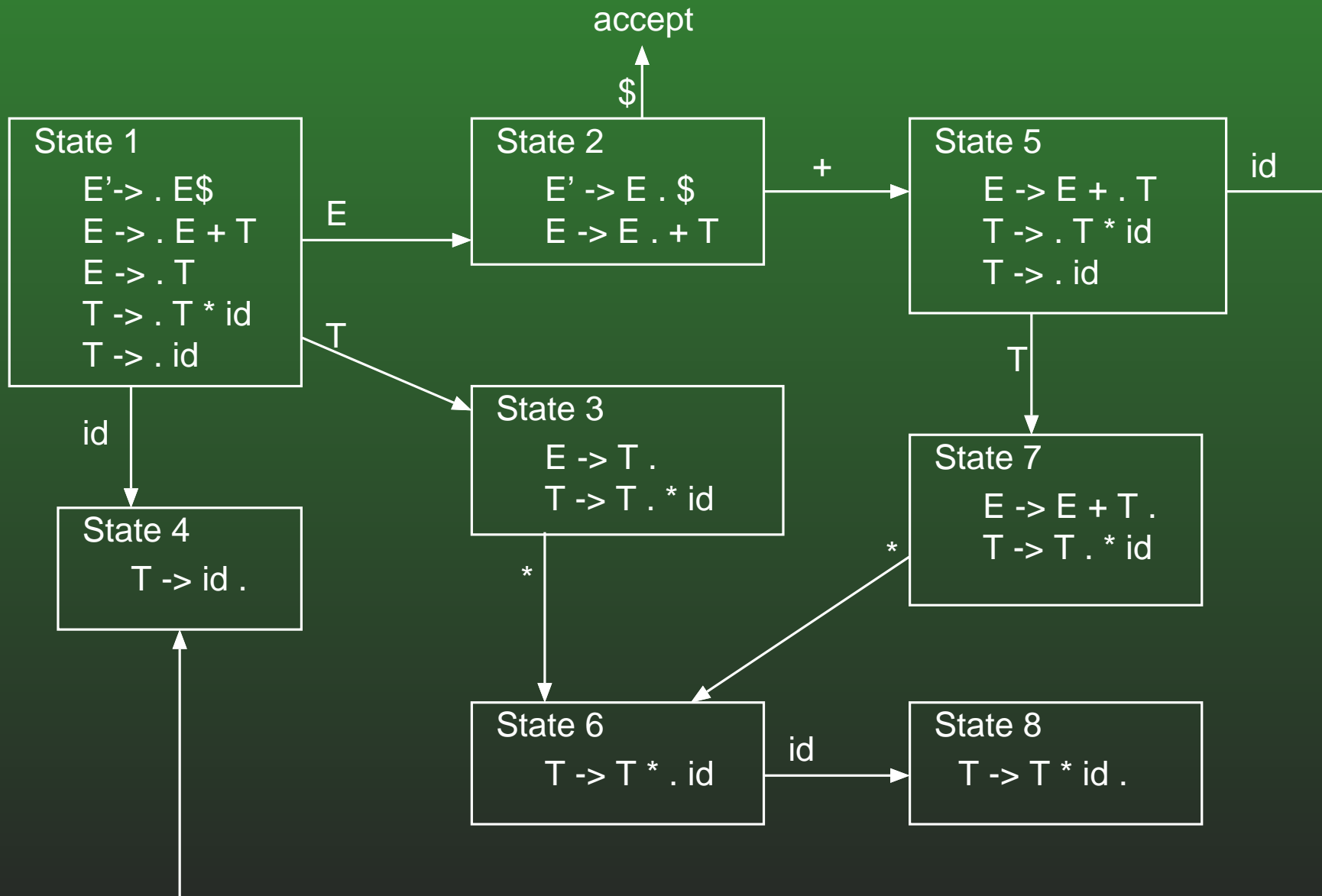
$$(1) E \rightarrow E + T$$

$$(2) E \rightarrow T$$

$$(3) T \rightarrow T * id$$

$$(4) T \rightarrow id$$

# 05-19: LR(0) States & Transitions



# 05-20: LR(0) Parse Table

---

	id	+	*	\$	$E$	$T$
1	s4				g2	g3
2		s5		accept		
3	r(2)	r(2)	r(2),s6	r(2)		
4	r(4)	r(4)	r(4)	r(4)		
5	s4					g7
6	s8					
7	r(1)	r(1)	r(1),s6	r(1)		
8	r(3)	r(3)	r(3)	r(3)		

## 05-21: Shift-Reduce Conflict

---

- In state 3, on a \*, should we shift, or reduce? Why?

## 05-22: Shift-Reduce Conflict

---

- In state 3, on a \*, should we shift, or reduce? Why?
  - If we reduce, then we're stuck – since the top of the stack will contain  $E$ , the next symbol in the input stream is \*, and \* cannot follow  $E$  in any partial derivation!
- If a state contains the item:

$$S \rightarrow \gamma .$$

we should only reduce if *the next terminal can follow  $S$*

## 05-23: SLR(1)

---

- Add simple lookahead (the  $S$  in SLR(1) is for *simple*)
- In LR(0) parsers, if state  $k$  contains the item “ $S \rightarrow \gamma \cdot$ ” (where  $S \rightarrow \gamma$  is rule  $(n)$ )
  - Put  $r(n)$  in state  $k$ , in all columns
- In SLR(0) parsers, if state  $k$  contains the item “ $S \rightarrow \gamma \cdot$ ” (where  $S \rightarrow \gamma$  is rule  $(n)$ )
  - Put  $r(n)$  in state  $k$ , in all columns *in the follow set of  $S$*

# 05-24: SLR(1) Parse Table

---

	id	+	*	\$	$E$	$T$
1	s4				g2	g3
2		s5		accept		
3		r(2)	s6	r(2)		
4		r(4)	r(4)	r(4)		
5	s4					g7
6	s8					
7		r(1)	s6	r(1)		
8		r(3)	r(3)	r(3)		

$x + y * z$

$x * y + z$

$w + x + y * z$

$w * x * y + z$

# 05-25: YALR(0)E

---

- Yet another LR(0) Example

$$(0) S' \rightarrow S\$$$

$$(1) S \rightarrow AB$$

$$(2) A \rightarrow Aa$$

$$(3) A \rightarrow a$$

$$(4) B \rightarrow Bb$$

$$(4) B \rightarrow b$$

05-26: **YALR(0)E**

---

# 05-27: YASLR(1)E

---

- Yet another SLR(1) Example

$$(0) E' \rightarrow E\$$$

$$(1) E \rightarrow V$$

$$(2) E \rightarrow \text{num}$$

$$(3) V \rightarrow \text{id}$$

$$(4) V \rightarrow V[E]$$

# 05-28: Yet Another Example

---

$$(0) S' \rightarrow S\$$$

$$(1) S \rightarrow L = R$$

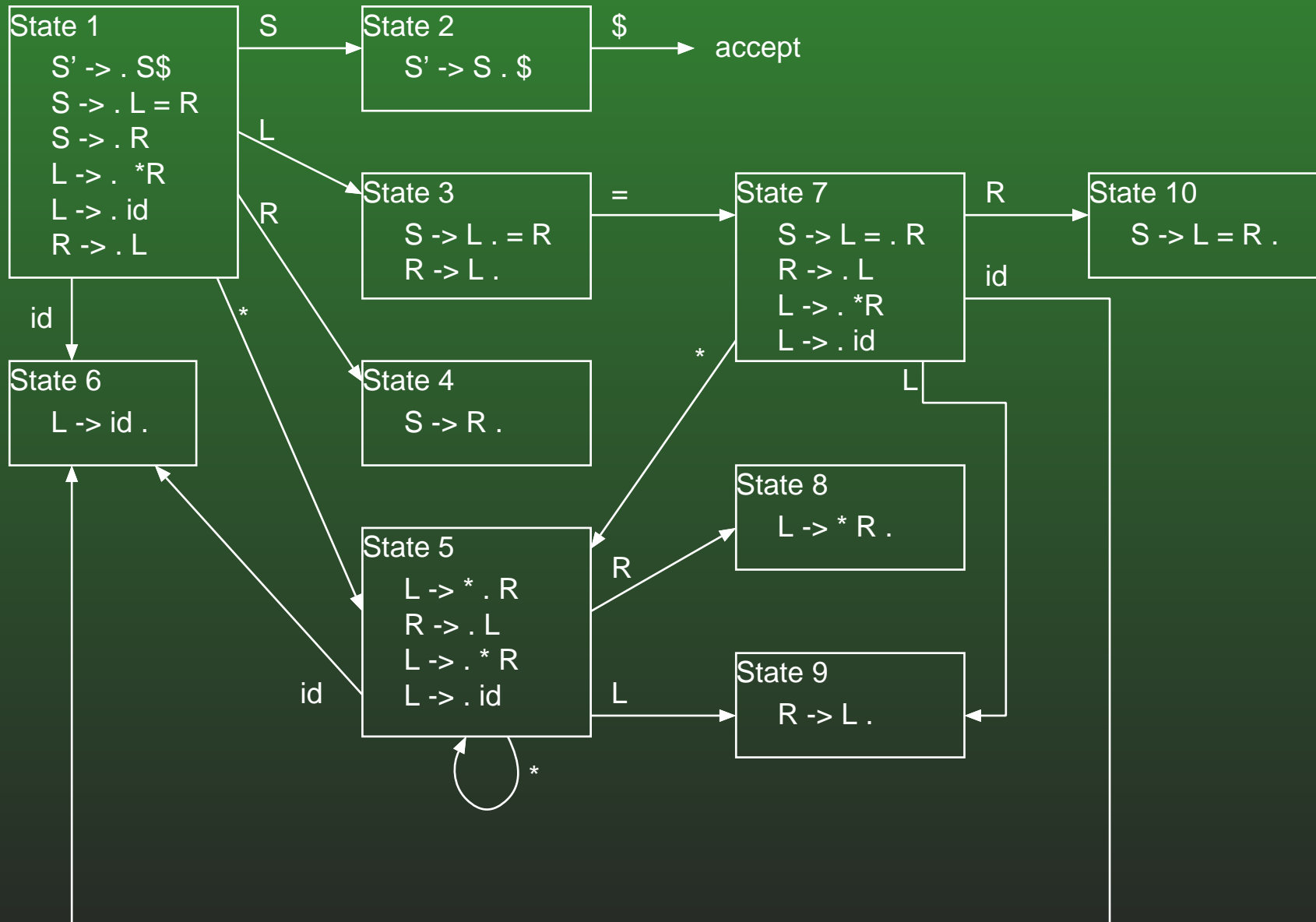
$$(2) S \rightarrow R$$

$$(3) L \rightarrow *R$$

$$(4) L \rightarrow \text{id}$$

$$(5) R \rightarrow L$$

# 05-29: LR(0) States & Transitions



# 05-30: SLR(1) Parse Table

---

	id	=	*	\$	$S$	$L$	$R$
1	s6		s5		g2	g3	g4
2				accept			
3		r(5),s7		r(5)			
4				r(2)			
5	s6		s5			g9	g8
6		r(4)		r(4)			
7	s6		s5			g9	g10
8		r(3)		r(3)			
9		r(5)		r(5)			
10				r(1)			

## 05-31: Why SLR(1) Fails

---

$S \rightarrow L \cdot = R$

$R \rightarrow L \cdot$

- In this state, on a =, should be shift or reduce?
- An = can follow an  $R$  – only if the  $R$  is preceded by  $a^*$
- We need a more sophisticated lookahead scheme to disambiguate this situation

## 05-32: LR(1) Items

---

- Like LR(0) items, contain a rule with a “.”
- Also contain lookahead information – the terminals that could follow this rule, *in the current derivation*
  - More sophisticated than SLR(1), which only look at what terminals could follow the LHS of the rule in *any* derivation

## 05-33: LR(1) Example

---

$$(0) S' \rightarrow S\$$$

$$(1) S \rightarrow L = R$$

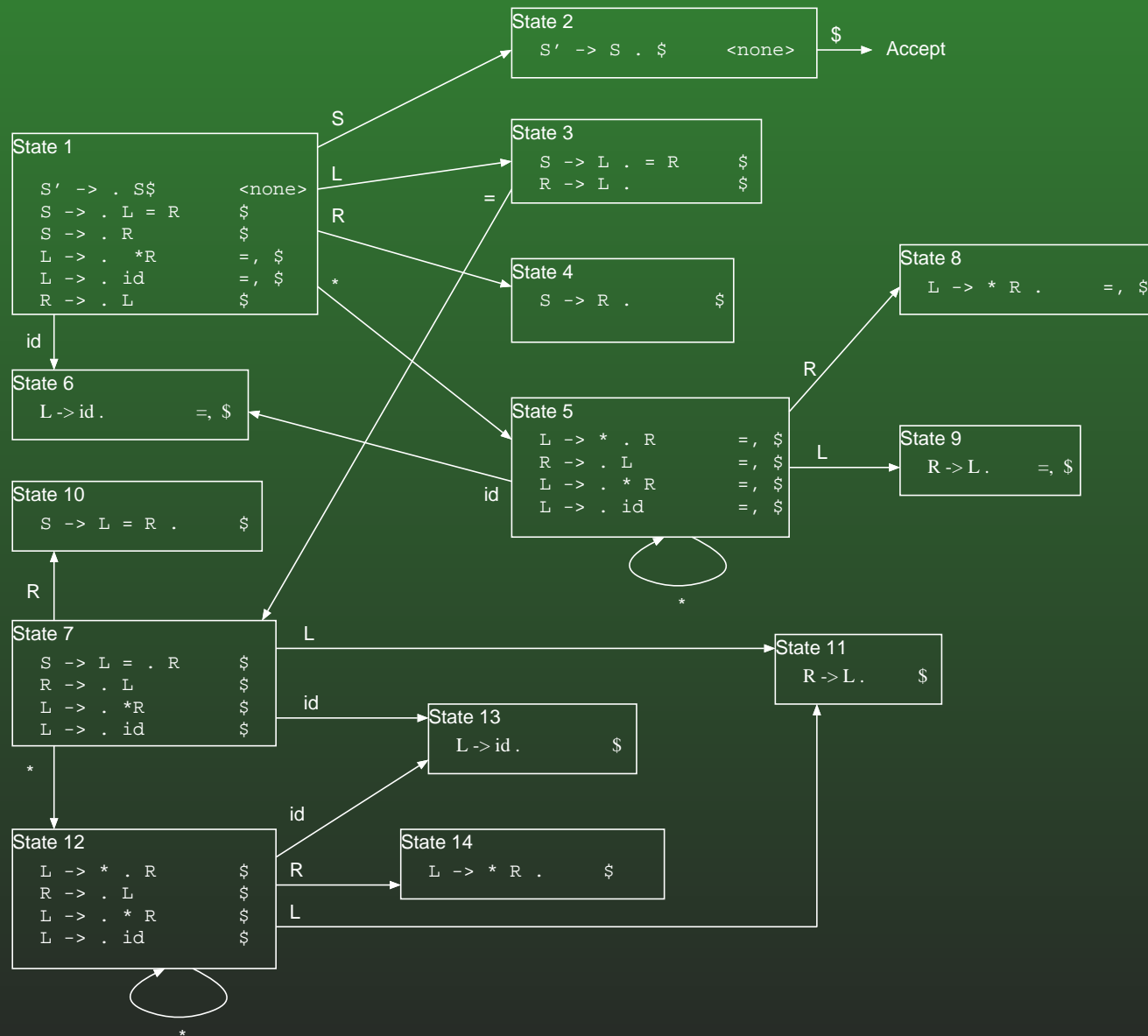
$$(2) S \rightarrow R$$

$$(3) L \rightarrow *R$$

$$(4) L \rightarrow \text{id}$$

$$(5) R \rightarrow L$$

# 05-34: LR(1) States and Transitions



# 05-35: LR(1) Parse Table

---

	id	=	*	\$	<i>S</i>	<i>L</i>	<i>R</i>
1	s6		s5		g2	g3	g4
2				accept			
3		s7		r(5)			
4				r(2)			
5	s6		s5			g9	g8
6		r(4)		r(4)			
7	s13		s12			g11	g10
8		r(3)		r(3)			
9		r(5)		r(5)			
10				r(1)			
11				r(5)			
12	s13		s12			g11	g14
13				r(4)			
14		r(3)		r(3)			

## 05-36: More LR(1) Examples

---

$$(0) S' \rightarrow S\$$$

$$(1) S \rightarrow BC$$

$$(2) S \rightarrow b$$

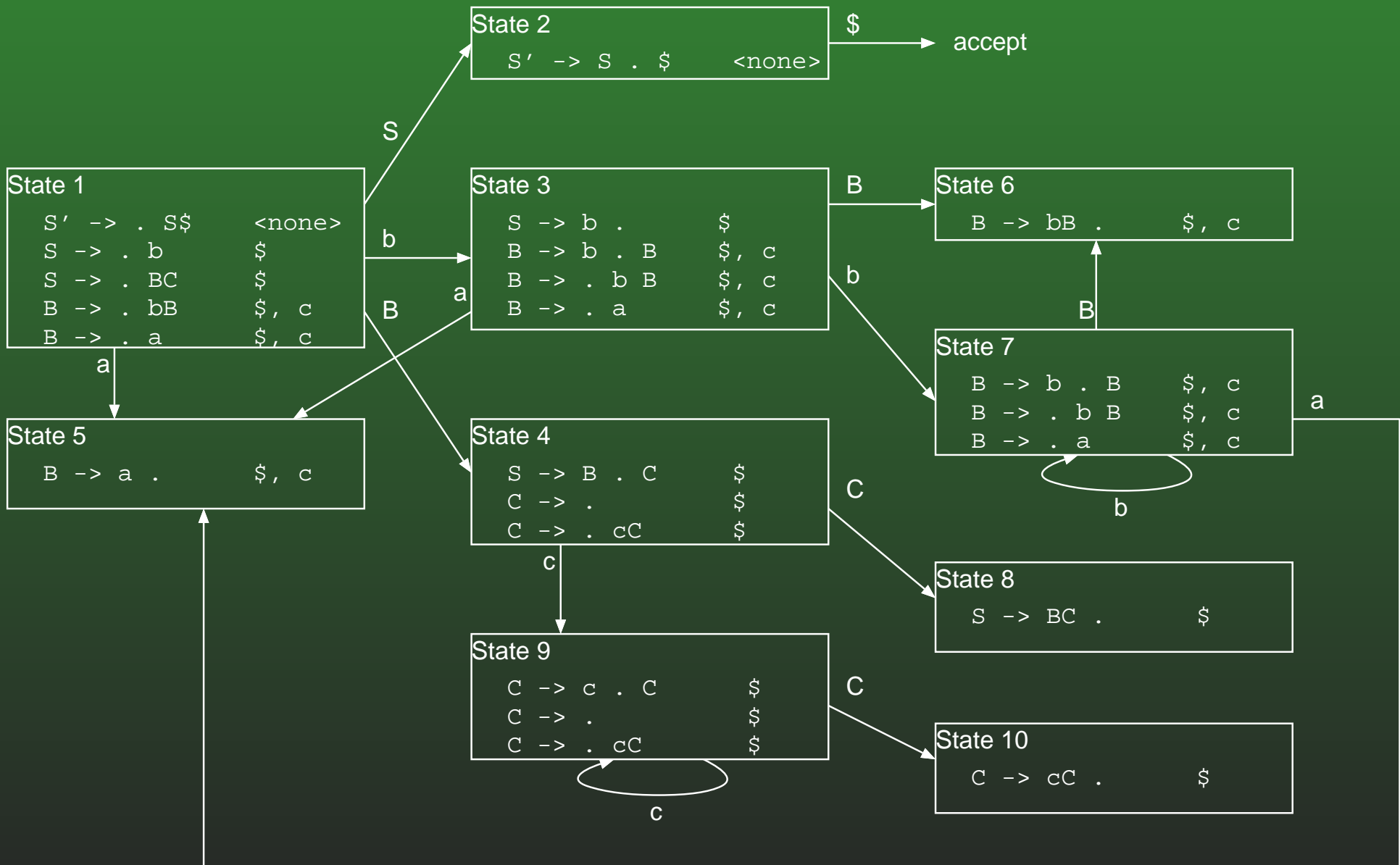
$$(3) B \rightarrow bB$$

$$(4) B \rightarrow a$$

$$(5) C \rightarrow \epsilon$$

$$(6) C \rightarrow cC$$

# 05-37: LR(1) States & Transitions



# 05-38: LR(1) Parse Table

---

	a	b	c	\$	<i>S</i>	<i>B</i>	<i>C</i>
1	s5	s3			g2	g4	
2				accept			
3	s5	s7		r(2)		g6	
4			s9	r(5)			g8
5			r(4)	r(4)			
6			r(3)	r(3)			
7	s5	s7				g6	
8				r(1)			
9			s9				g10
10				r(6)			

## 05-39: LALR Parsers

---

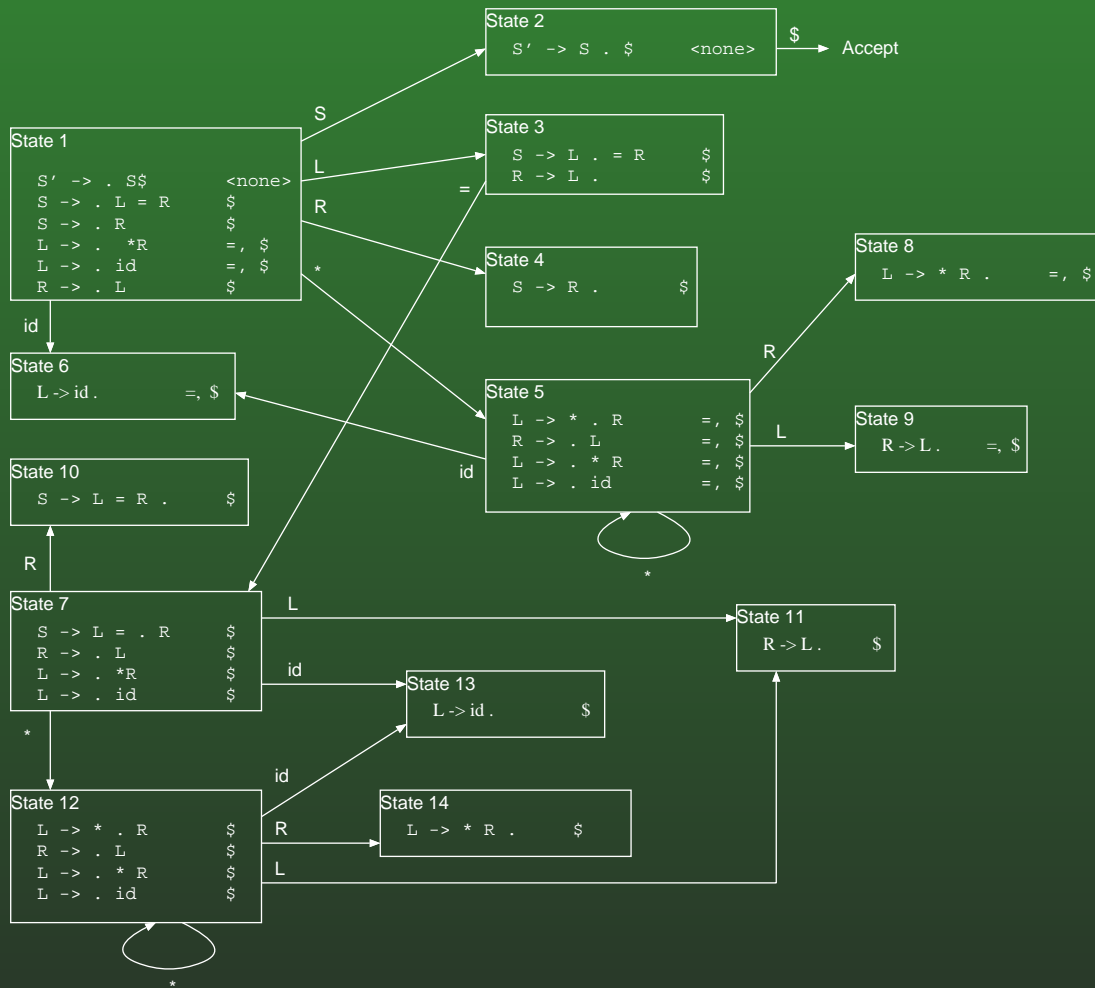
- LR(1) Parsers are more powerful than LR(0) or SLR(1) parsers
- LR(1) Parsers can have many more states than LR(0) or SLR(1) parsers
  - My simpleJava implementation has 139 LR(0) states, and *thousands* of LR(1) states
- We'd like *nearly* the power of LR(1), with the memory requirements of LR(0)

## 05-40: LALR Parsers

---

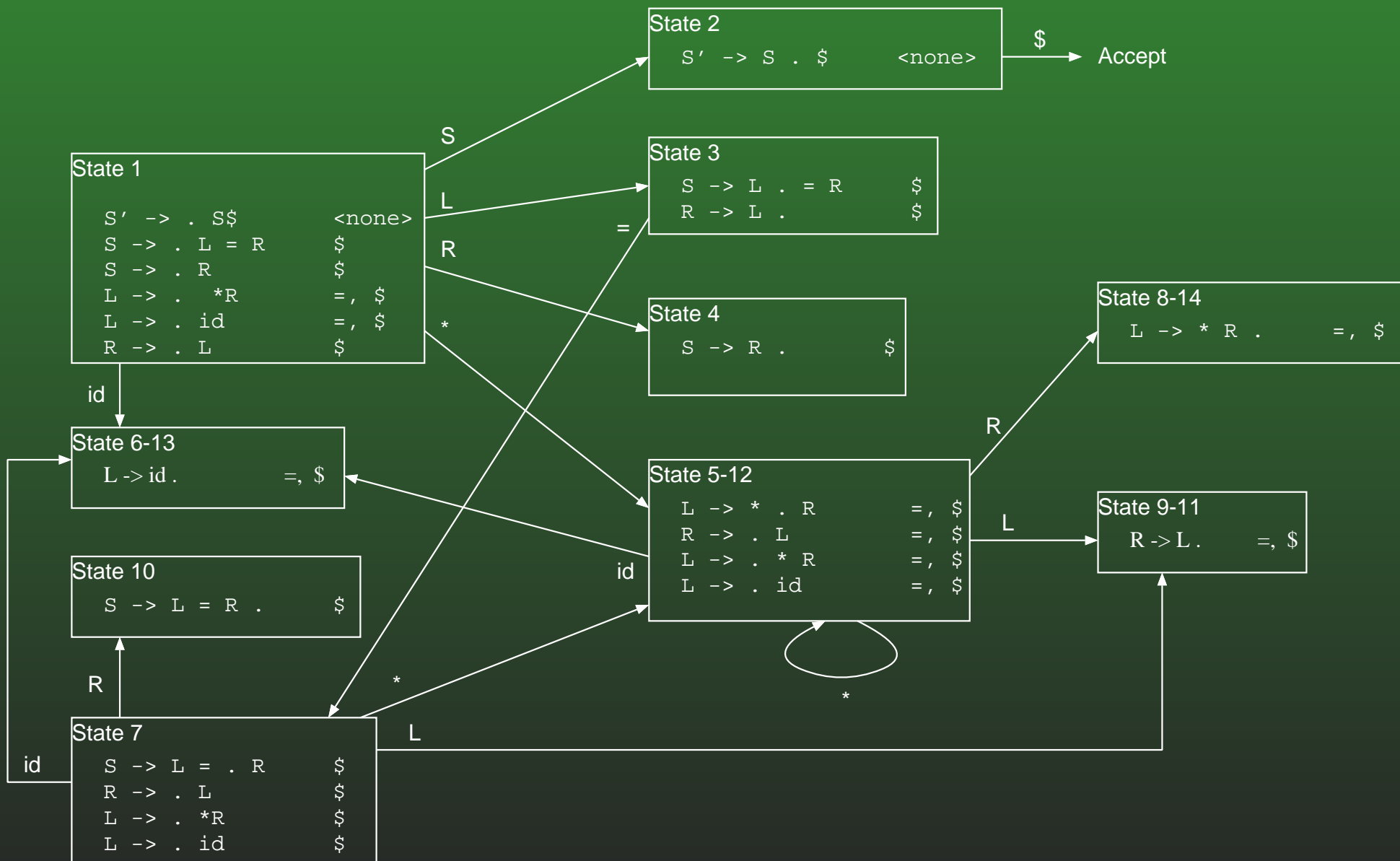
- LR(1) parsers can have large numbers of states
- Many of the states will be nearly the same – they will differ only in Lookahead
- IDEA – Combine states that differ only in lookahead values
  - Set lookahead of combined state to union of lookahead values from combining states

# 05-41: LALR Parser Example



Can combine 5 & 12, 6 & 13, 8 & 14, 9 & 11

# 05-42: LALR Parser Example



# 05-43: LALR Parser Example

---

	id	=	*	\$	<i>S</i>	<i>L</i>	<i>R</i>
1	s6-13		s5-12		g2	g3	g4
2				accept			
3		s7		r(5)			
4				r(2)			
5-12	s6-13		s5-12			g9-11	g8-14
6-13		r(4)		r(4)			
7	s6-13		s5-12			g9-11	g10
8-14		r(3)		r(3)			
9-11		r(5)		r(5)			
10				r(1)			

# 05-44: More LALR Examples

---

$$(0) S' \rightarrow S\$$$

$$(1) S \rightarrow Aa$$

$$(2) S \rightarrow bAc$$

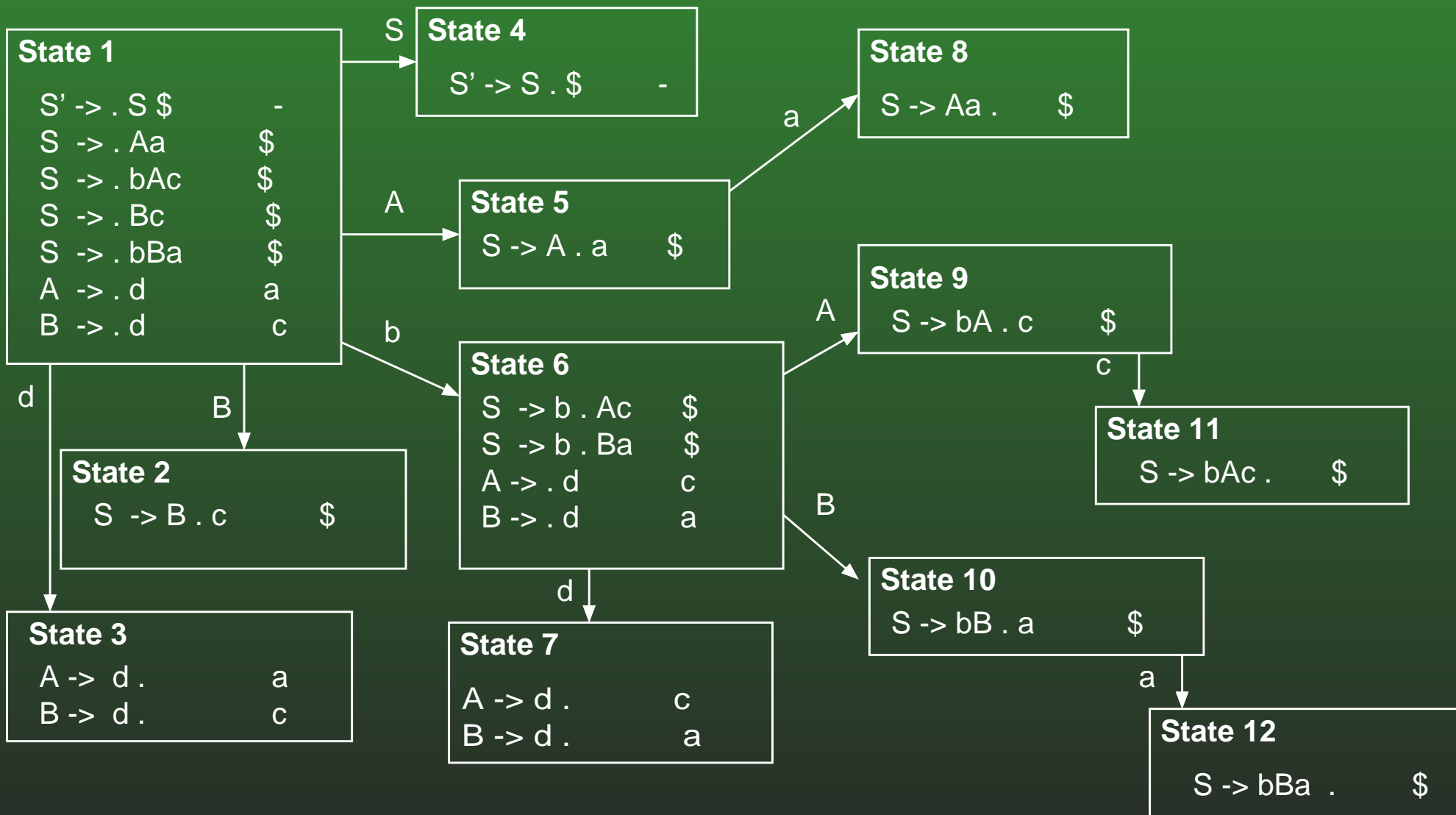
$$(3) S \rightarrow Bc$$

$$(4) S \rightarrow bBa$$

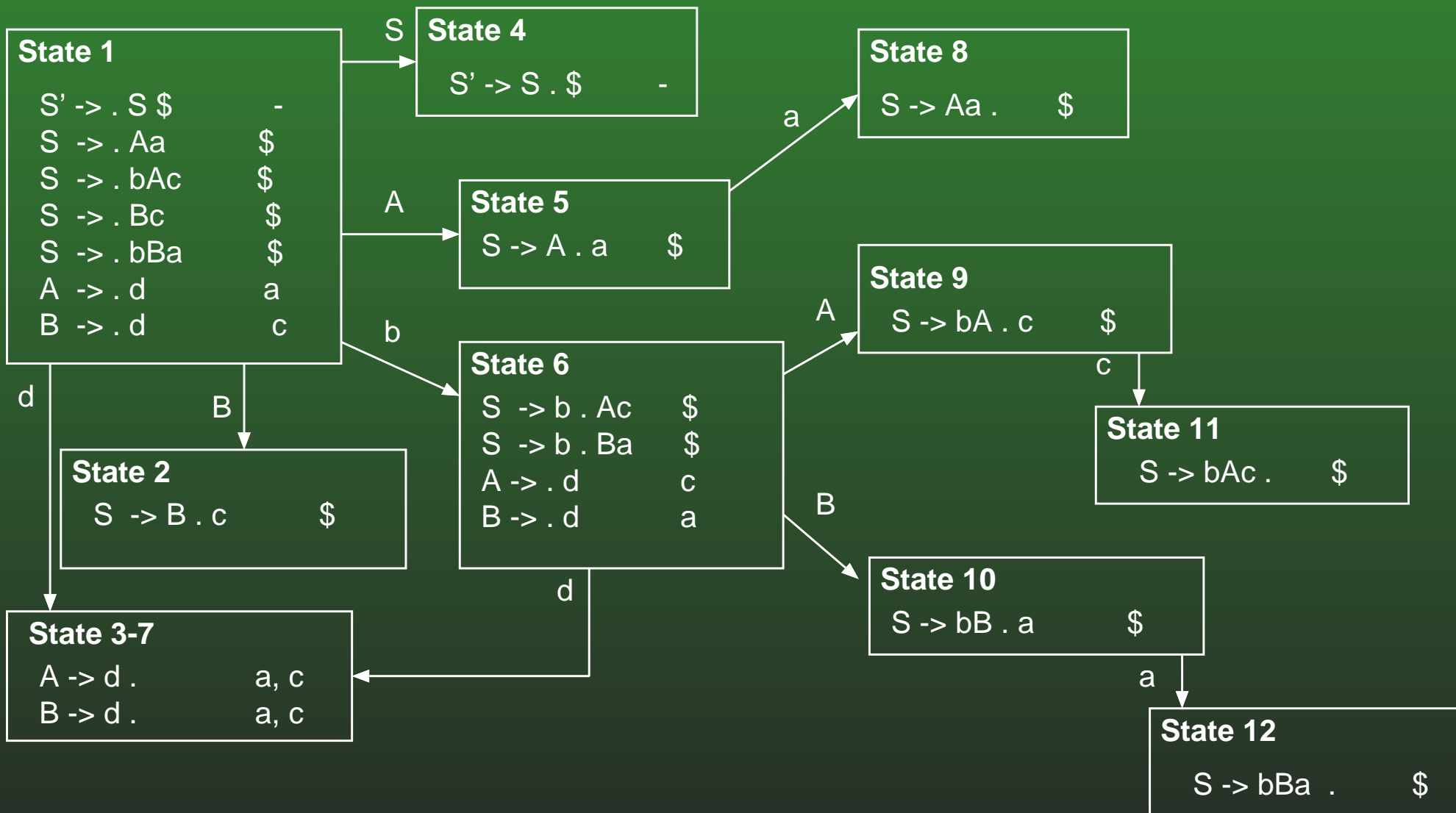
$$(5) A \rightarrow d$$

$$(6) B \rightarrow d$$

# 05-45: More LALR Examples



# 05-46: More LALR Examples



# 05-47: More LALR Examples

