Sub: Compiler Construction
Syntax Analysis PART 5

Compiled for: 7th Sem, CE, DDU

Compiled by: Niyati J. Buch

Ref. : Compilers: Principles, Techniques, and Tools, 2nd Edition
Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman

Topics Covered

- Bottom-up parsing
  - CLR
    - Constructing LR(1) item set
    - Constructing LR(1) parsing table
  - LALR
    - LALR(1) item sets
    - LALR(1) parsing table
    - Example to demonstrate reduce/reduce conflict after merger
    - Show that the grammar is LALR(1) but not SLR(1)
    - Show that the grammar is LR(1) but not LALR(1)
  - Ambiguous grammar
    - Expression grammar
    - Dangling else grammar
  - Error Recovery

More Powerful LR Parsers

1. The **"canonical-LR"** or just "LR" method, which makes full use of the lookahead symbol(s).
   - This method uses a large set of items, called the **LR(1) items**.
2. The **"lookahead-LR"** or "LALR" method, which is based on the **LR(0) sets of items**, and has many fewer states than typical parsers based on the LR(1) items.
   - By carefully introducing lookaheads into the LR(0) items, we can handle many more grammars with the LALR method than with the SLR method, and build parsing tables that are no bigger than the SLR tables.
   - **LALR is the method of choice in most situations.**

LR(1) Item

- The extra information is incorporated into the state by redefining items to include a terminal symbol as a second component. The general form of an item becomes **[A → α·β, a],** where **A → αβ** is a production and **a is a terminal or the right endmarker $**. We call such an object **an LR(1) item**. The 1 refers to the length of the second component, called the lookahead of the item.
- The lookahead has no effect in an item of the form [A → α·β, a] where β is not ∈, but an item of the form [A → α·, a] calls for a reduction by A → α only if the next input symbol is a. Thus, we are compelled to reduce by A → α only on those input symbols a for which [A → α·, a] is an LR(1) item in the state on top of the stack.

Constructing LR(1) Sets of Items

SetOfItems CLOSURE(I) {
    repeat
        for (each item $[A \to \alpha \cdot B\beta, a]$ in I)
            for (each production $B \to \gamma$ in G')
                for (each terminal b in FIRST(βa))
                    add $[B \to \cdot\gamma, b]$ to set I;
    until no more items are added to I;
    return I;
}

Constructing LR(1) Sets of Items

SetOfItems GOTO(I, X) {
    initialize J to be the empty set;
    for (each item $[A \to \alpha \cdot X \beta, a]$ in I)
        add item $[A \to \alpha X \cdot \beta, a]$ to set J;
    return CLOSURE(J);
}

Constructing LR(1) Sets of Items

```
void items(G') {
    initialize C to {CLOSURE({[S' → ·S, $]})};
    repeat
        for ( each set of items I in C )
            for ( each grammar symbol X )
                if ( GOTO(I, X) is not empty and not in C )
                    add GOTO(I, X) to C;
    until no new sets of items are added to C;
}
```

Constructing LR(1) Sets of Items

- Consider the following augmented grammar.

  S' → S

  S → C C

  C → c C | d
- The general form of an item becomes **[A → α·β, a],**

  where **A → αβ** is a production and **a is a terminal or the right endmarker $**
- For all productions B → γ add [B→ · γ, b] where b is FIRST(βa)

Constructing LR(1) Sets of Items

- Consider the following augmented grammar.

  S' → S

  S → C C

  C → c C | d
- The general form of an item becomes **[A → α·β, a],**

  where **A → αβ** is a production and **a is a terminal or the right endmarker $**
- [S' → · S, $] compare with **[A → α·Bβ, a],** A is S', α is ∈, B is S, β is ∈ and a is $
- For all productions B → γ i.e. S → CC, add [B→ · γ, b] where b is FIRST(βa) = FIRST(∈$) = {$}. Hence, [S → · CC, $] is added.
- [S → · CC, $] compare with **[A → α·Bβ, a],** A is S, α is ∈, B is C, β is C and a is $
- For all productions B → γ i.e C → cC|d, add [B→ · γ, b] where b is FIRST(βa) = FIRST(C$) = {c,d}. Hence, [C → · cC, c/d] and [C → · d, c/d] are added.

Constructing LR(1) Sets of Items

- Consider the following augmented grammar.

  S' → S

  S → C C

  C → c C | d

- The general form of an item becomes **[A → α·β, a],**

  where **A → αβ** is a production and **a is a terminal or the right endmarker $**

- [S' → · S, $] compare with **[A → α·Bβ, a],** A is S', α is ∈, B is S, β is ∈ and a is $
- For all productions B → γ i.e. S → CC, add [B→ · γ, b] where b is FIRST(βa) = FIRST(∈$) = {$}. Hence, [S → · CC, $] is added.
- [S → · CC, $] compare with **[A → α·Bβ, a],** A is S, α is ∈, B is C, β is C and a is $
- For all productions B → γ i.e C → cC|d, add [B→ · γ, b] where b is FIRST(βa) = FIRST(C$) = {c,d}. Hence, [C → · cC, c/d] and [C → · d, c/d] are added.

I_0

S' → · S ,$

S → · C C ,$

C → · c C ,c/d

C → · d, c/d

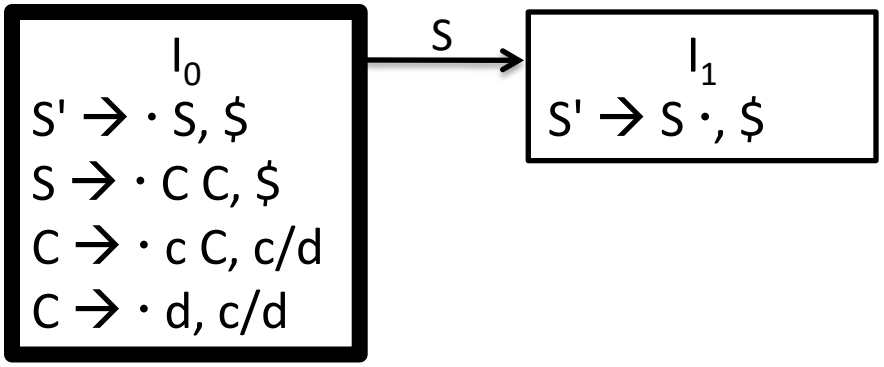
I_0

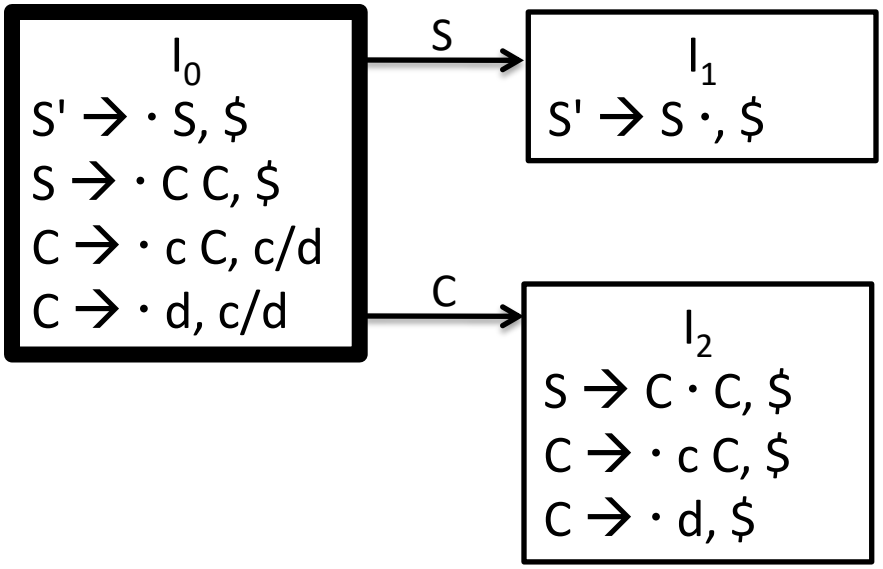
S' → · S, $

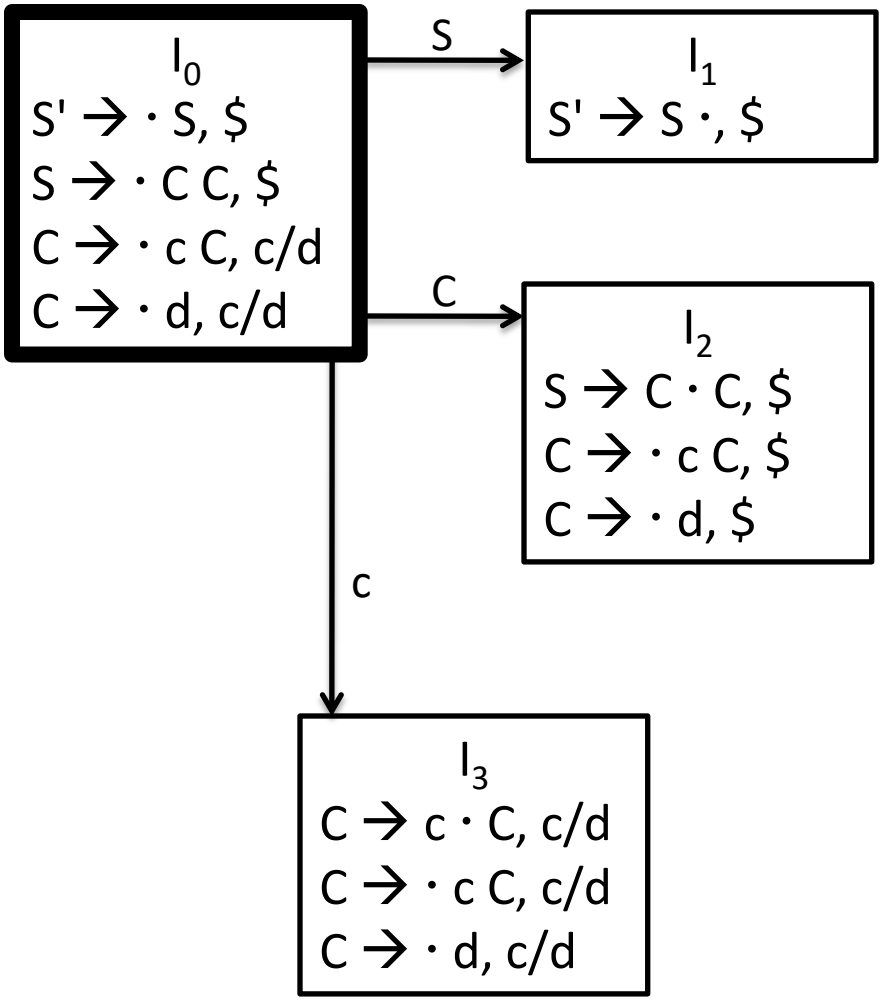
S → · C C, $

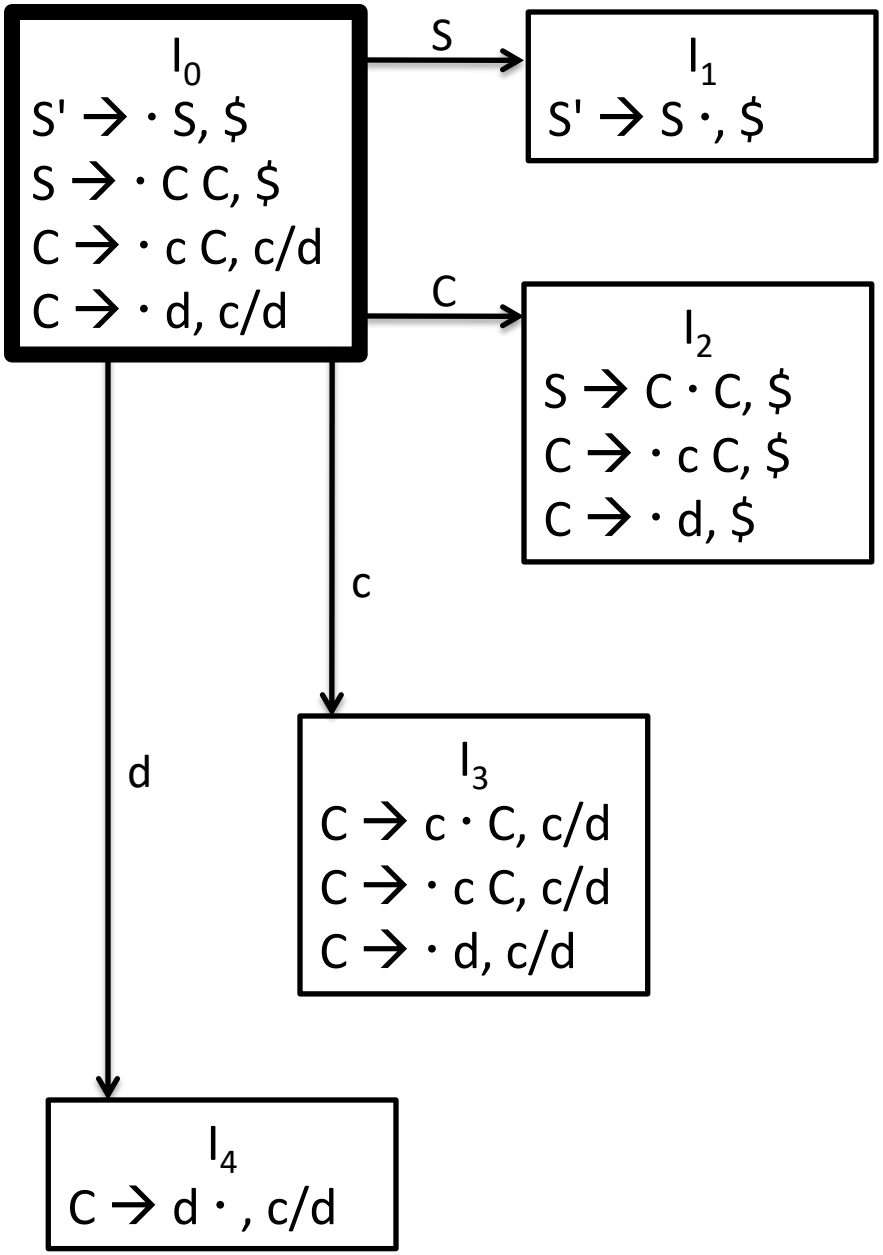
C → · c C, c/d

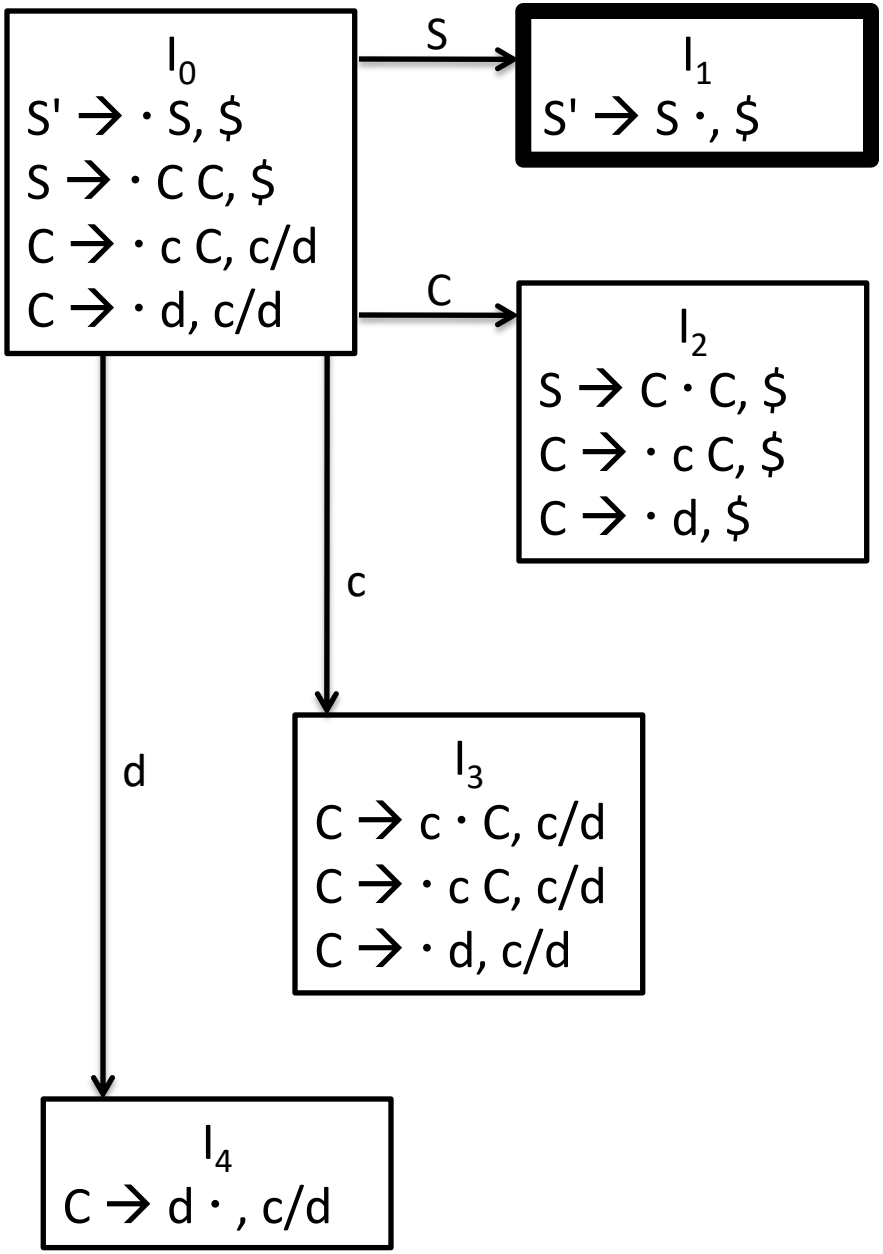
C → · d, c/d

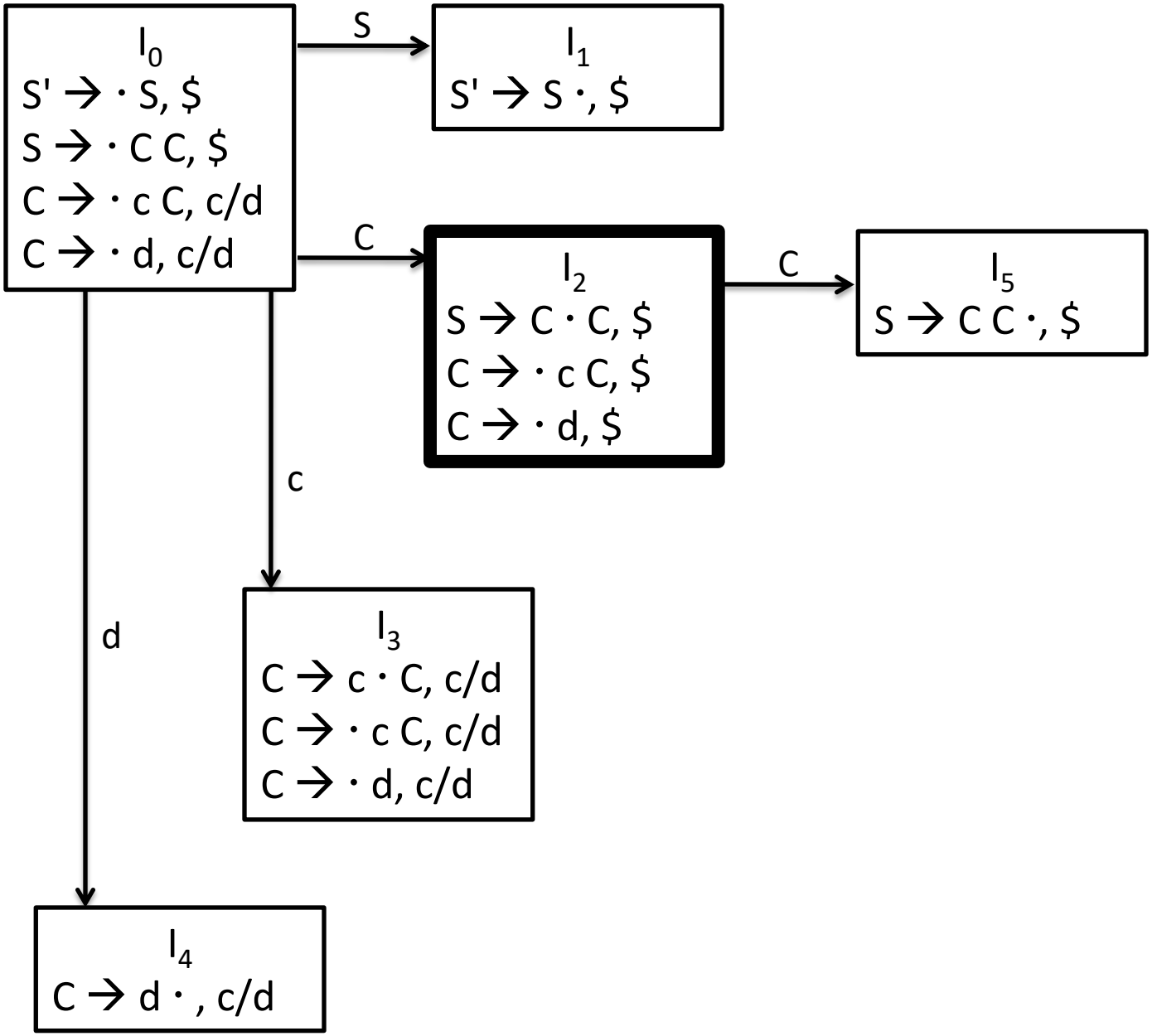

I_0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I_1
S' → S ·, $

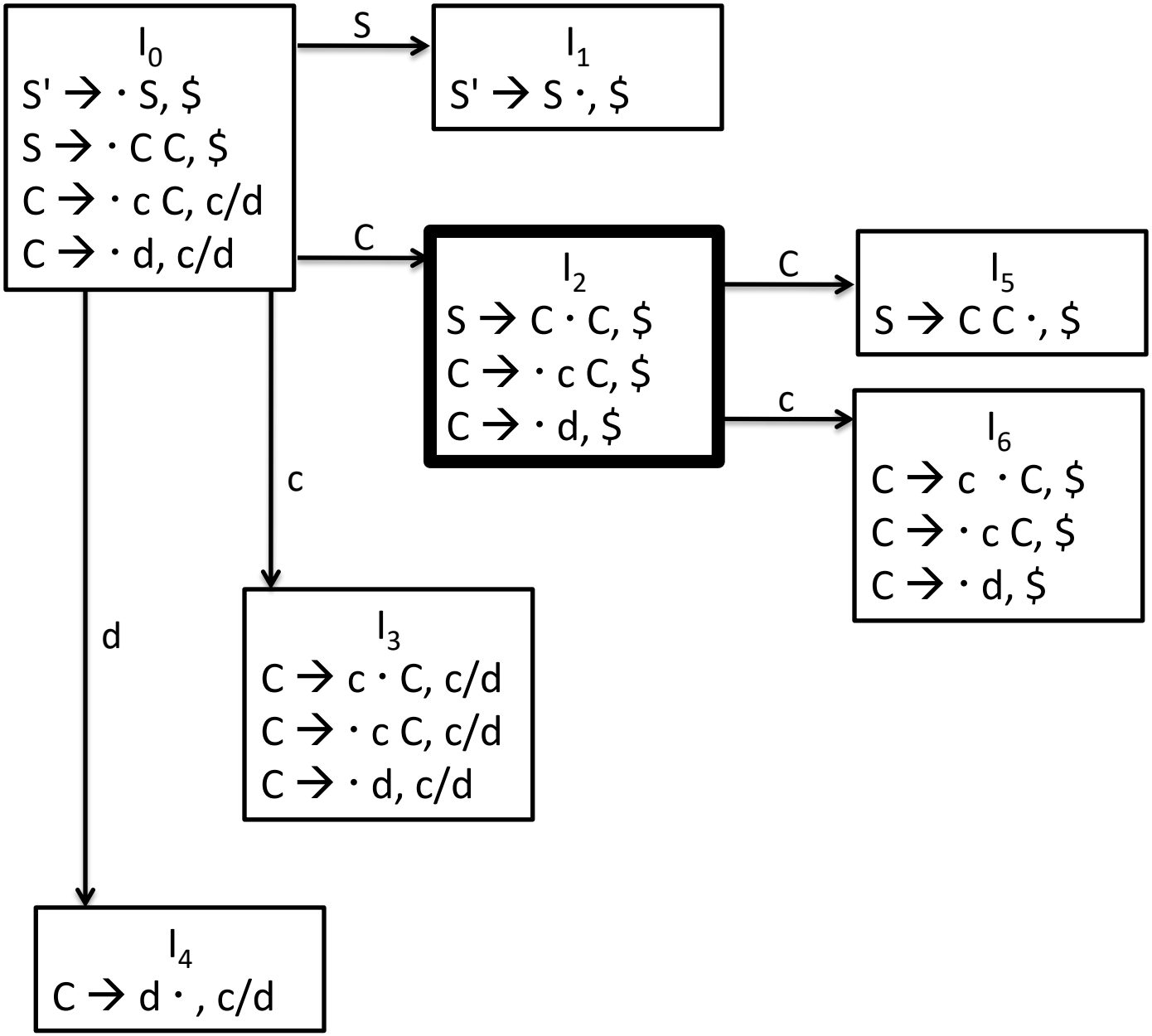

I_0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I_1
S' → S ·, $
C
I_2
S → C · C, $
C → · c C, $
C → · d, $

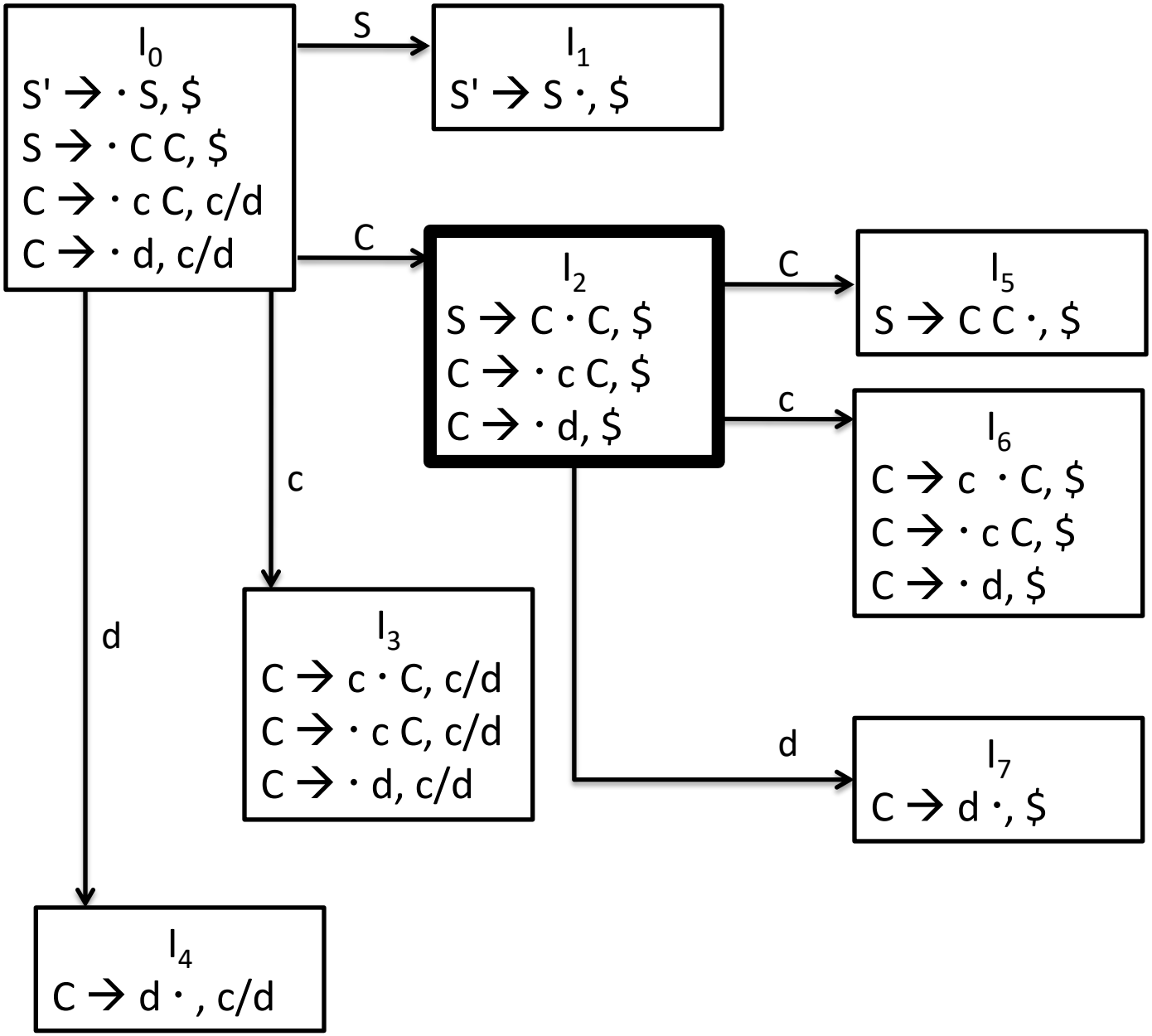

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d

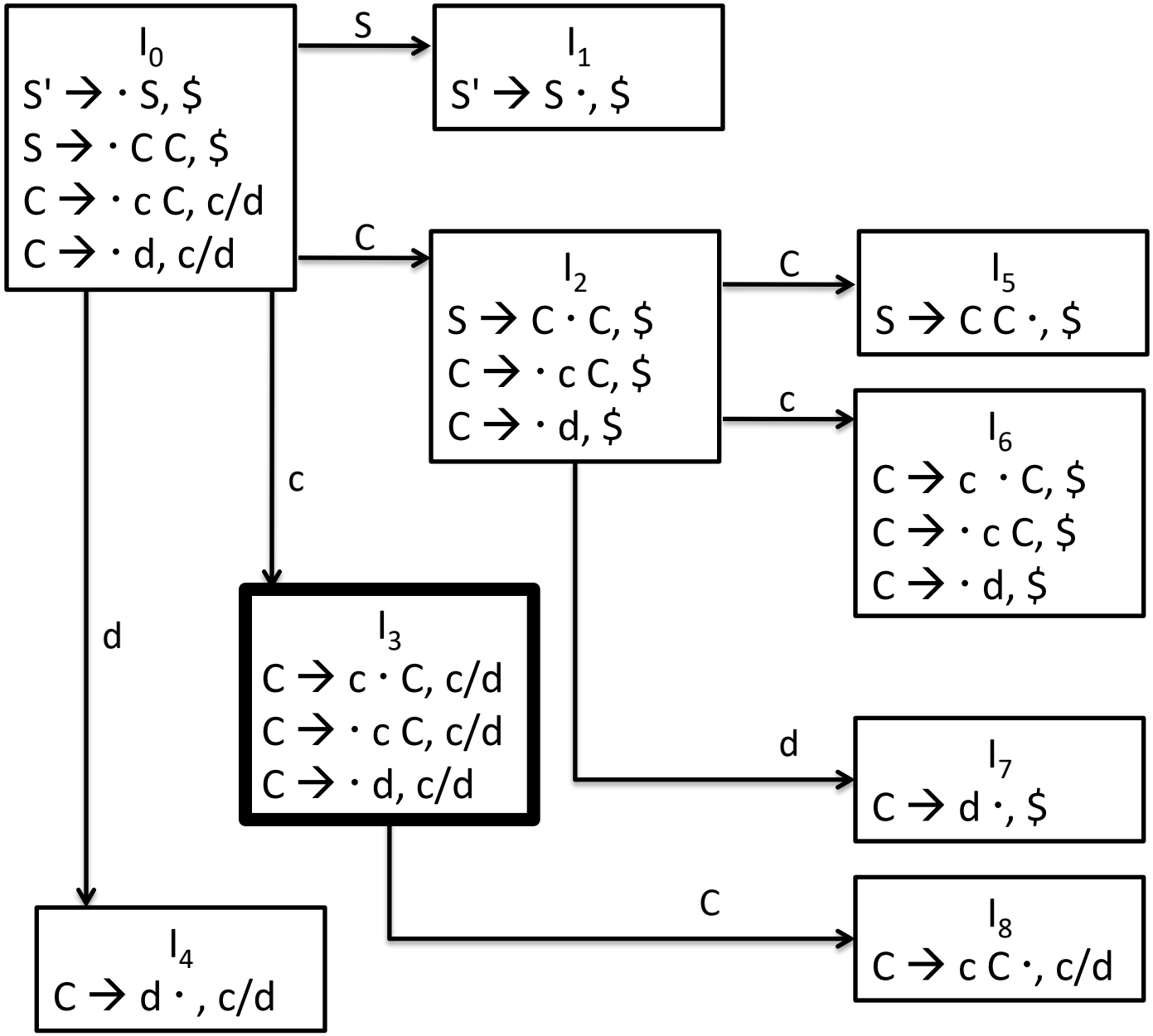

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
I4
C → d · , c/d

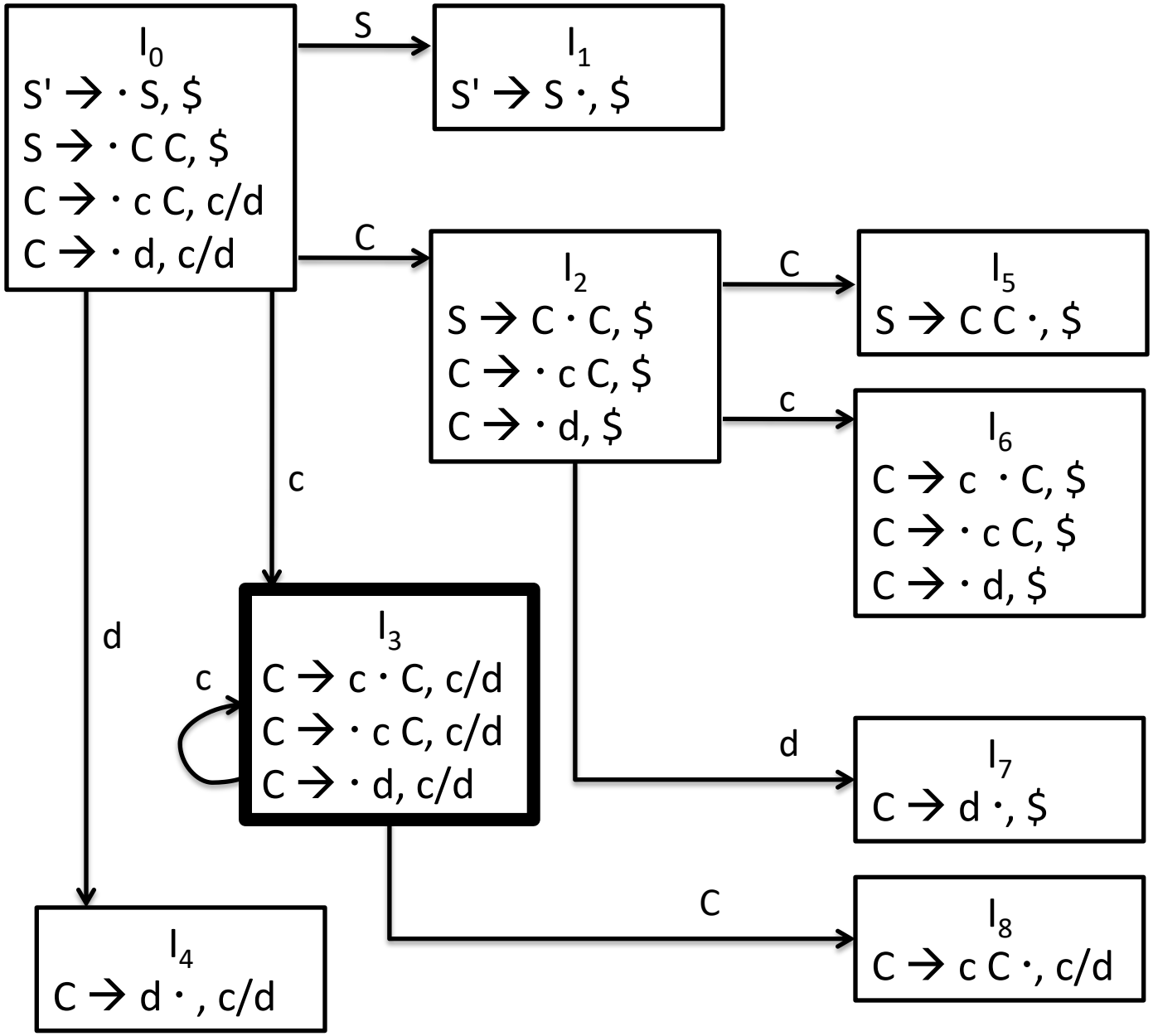

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
I4
C → d · , c/d

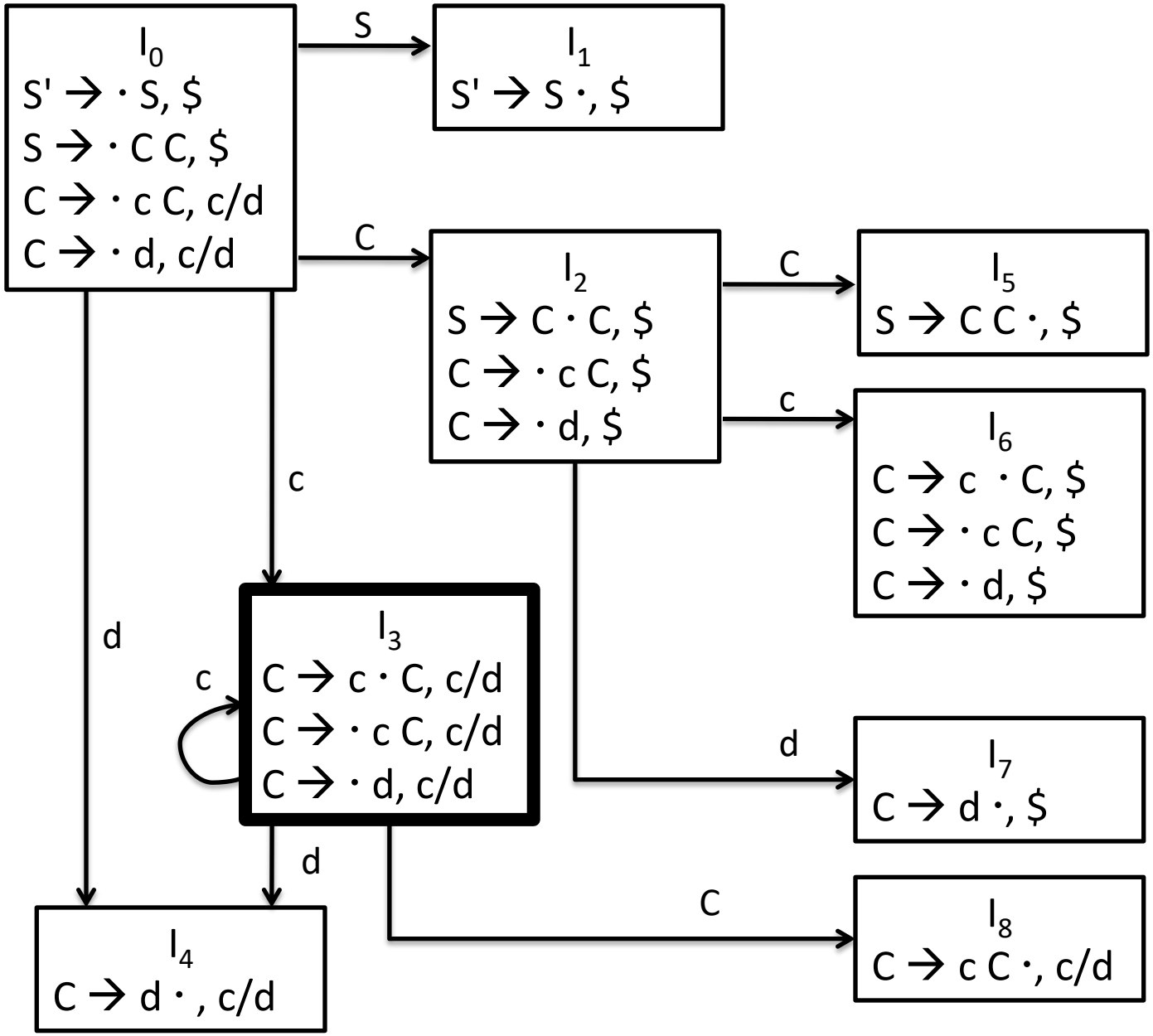

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
d
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
I4
C → d · , c/d

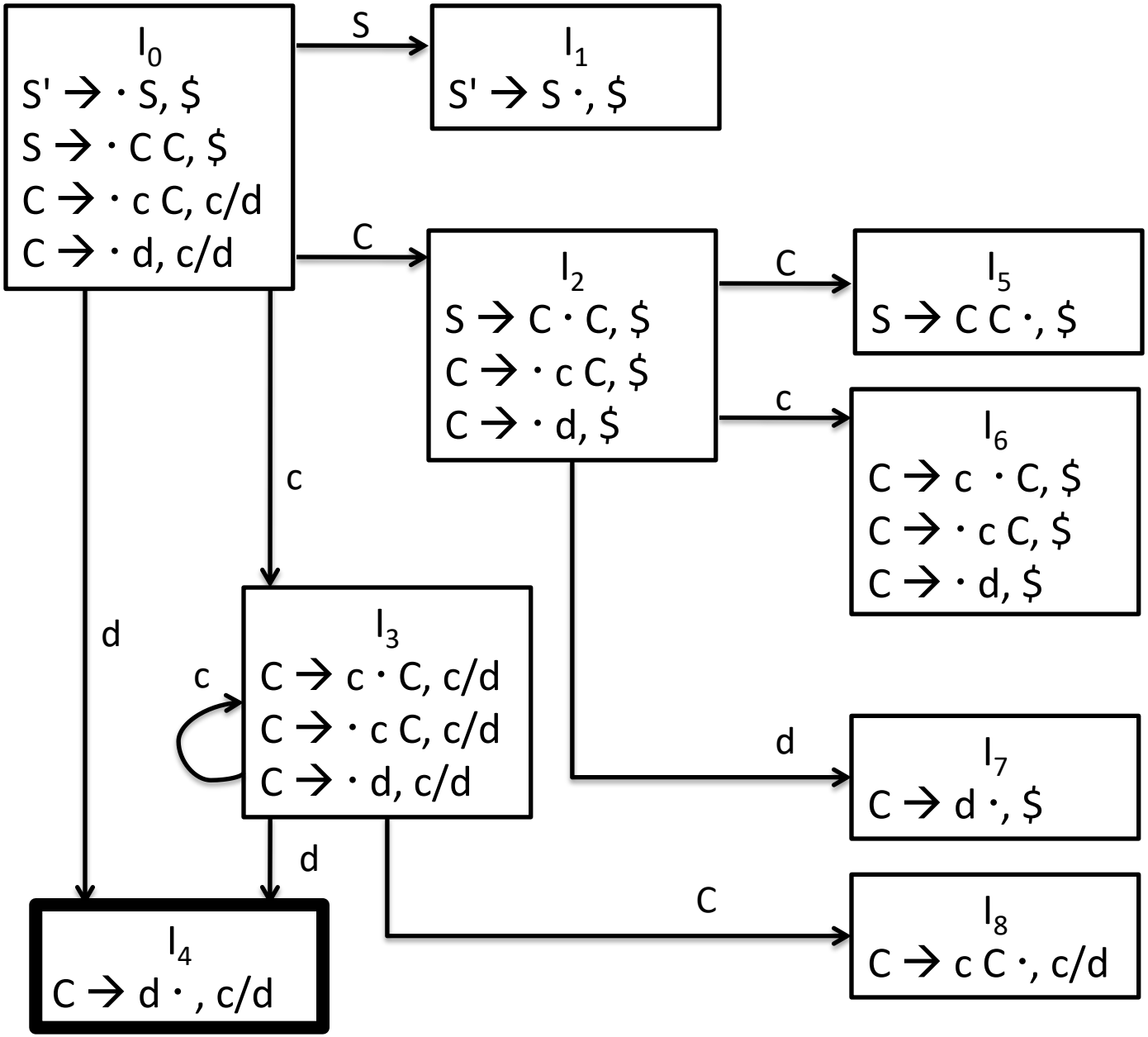

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
I4
C → d · , c/d

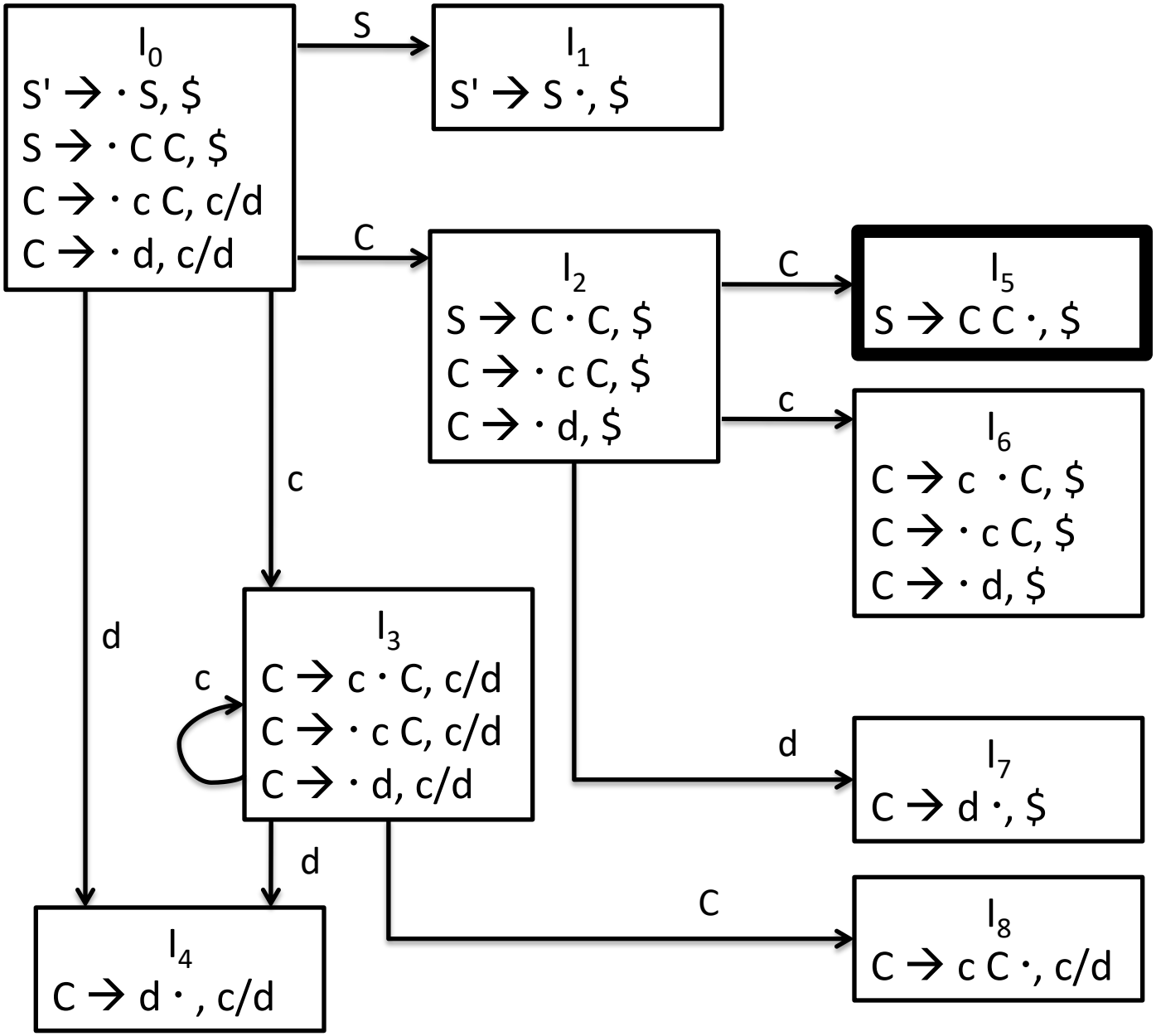

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
I4
C → d · , c/d
d
I7
C → d ·, $

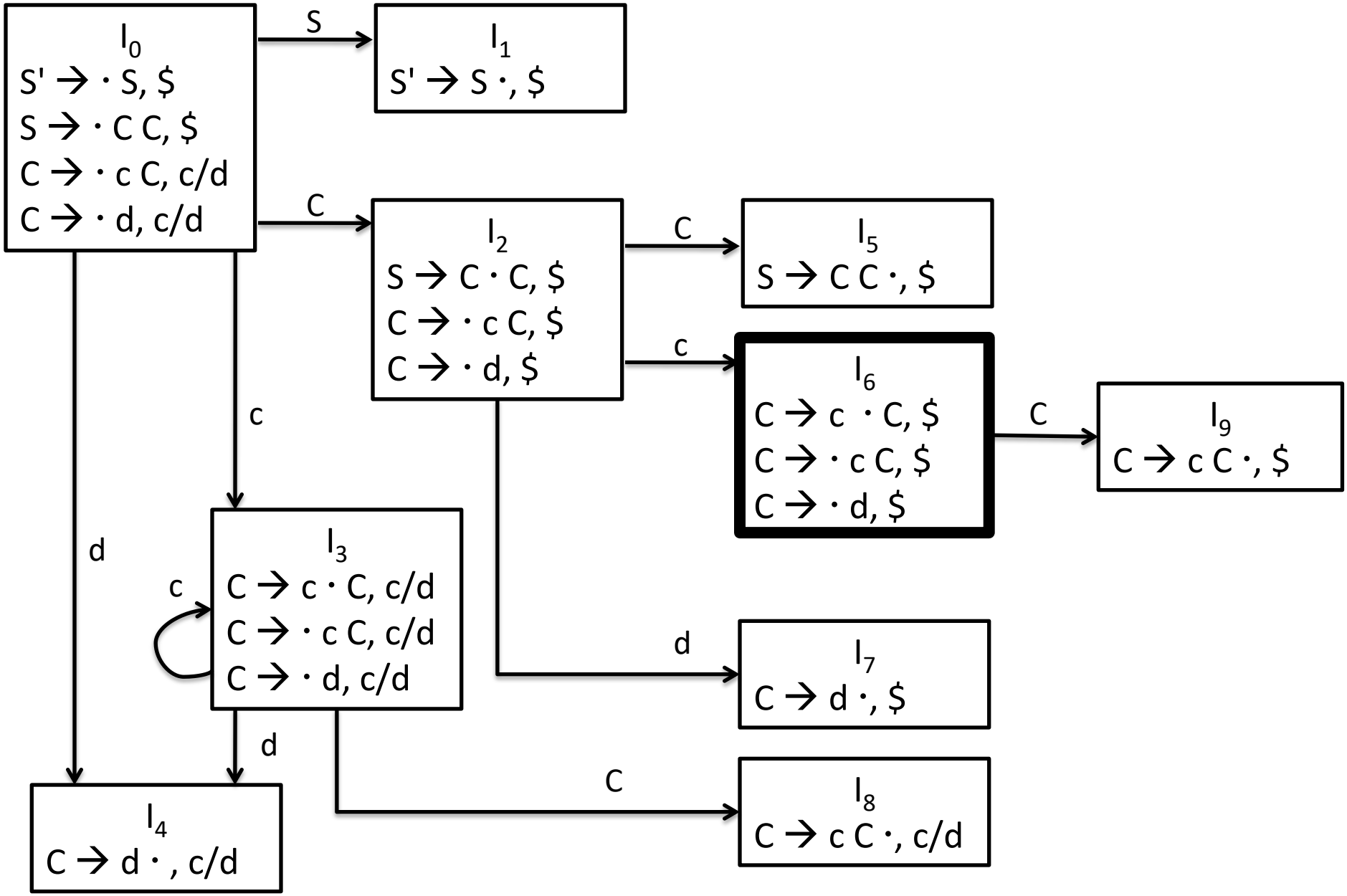

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
d
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
I7
C → d ·, $
C
I8
C → c C ·, c/d
I4
C → d · , c/d

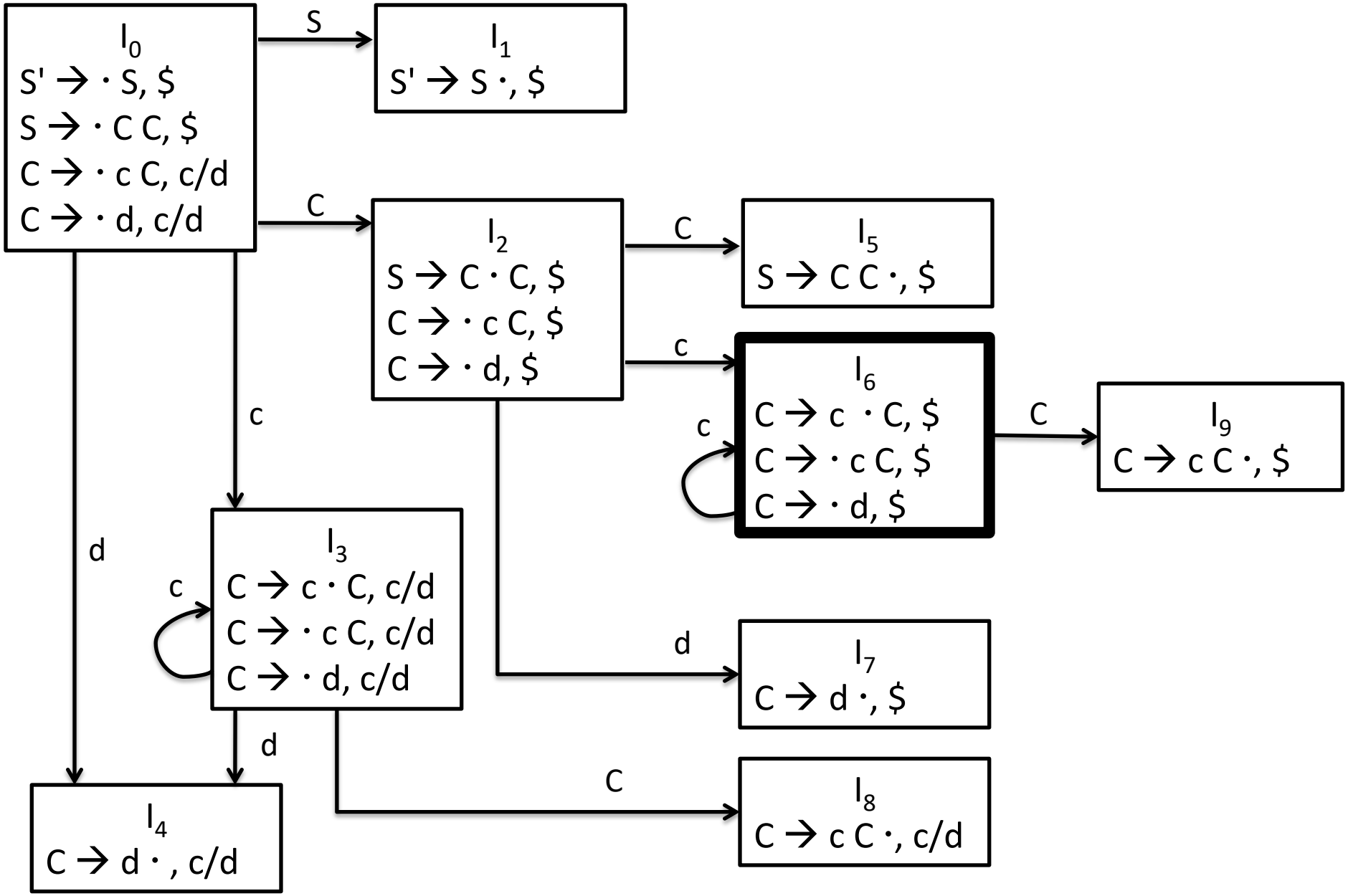

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
d
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
I7
C → d ·, $
C
I8
C → c C ·, c/d
I4
C → d · , c/d

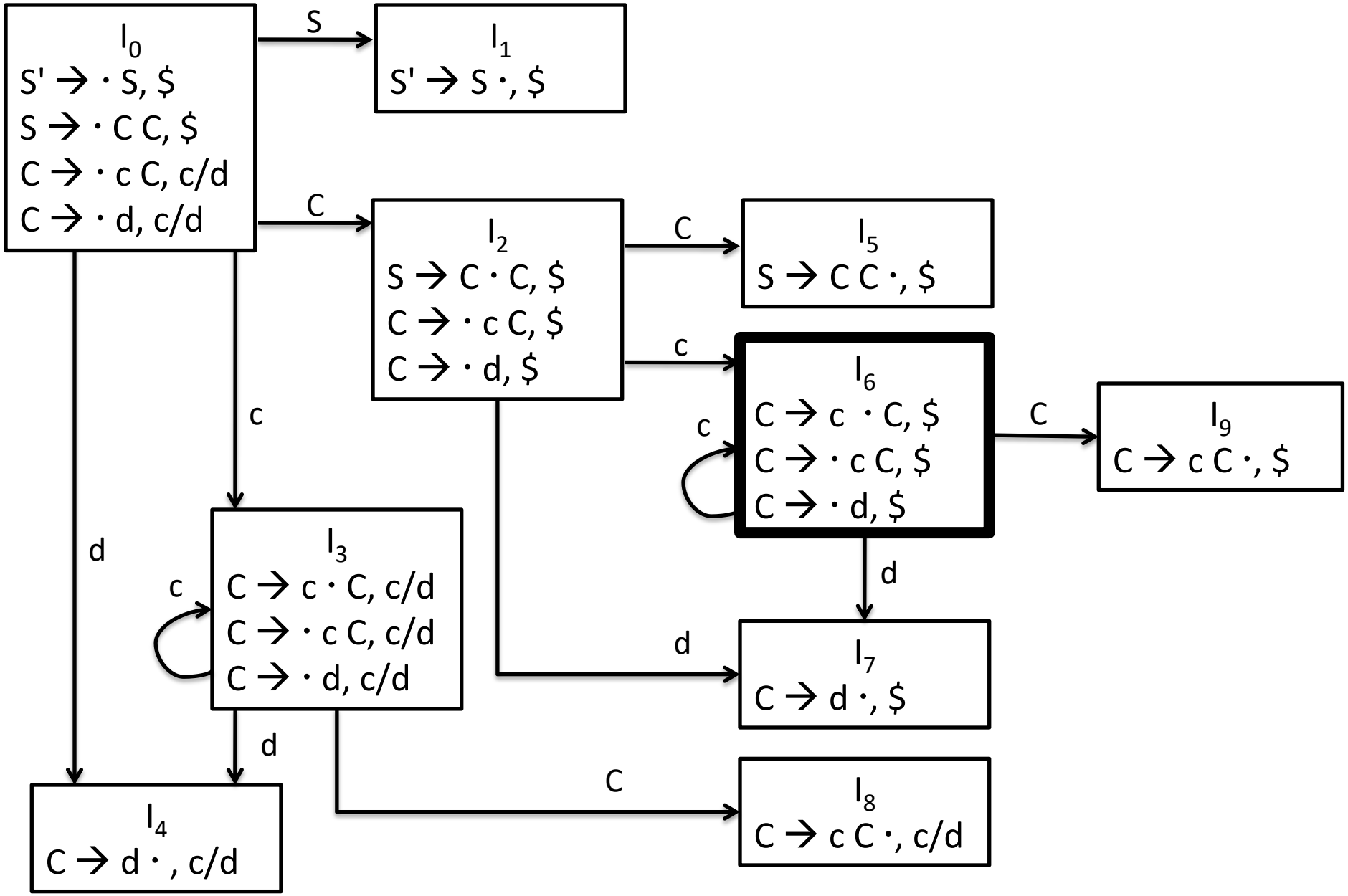

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
d
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
I7
C → d ·, $
d
C
I4
C → d · , c/d
I8
C → c C ·, c/d

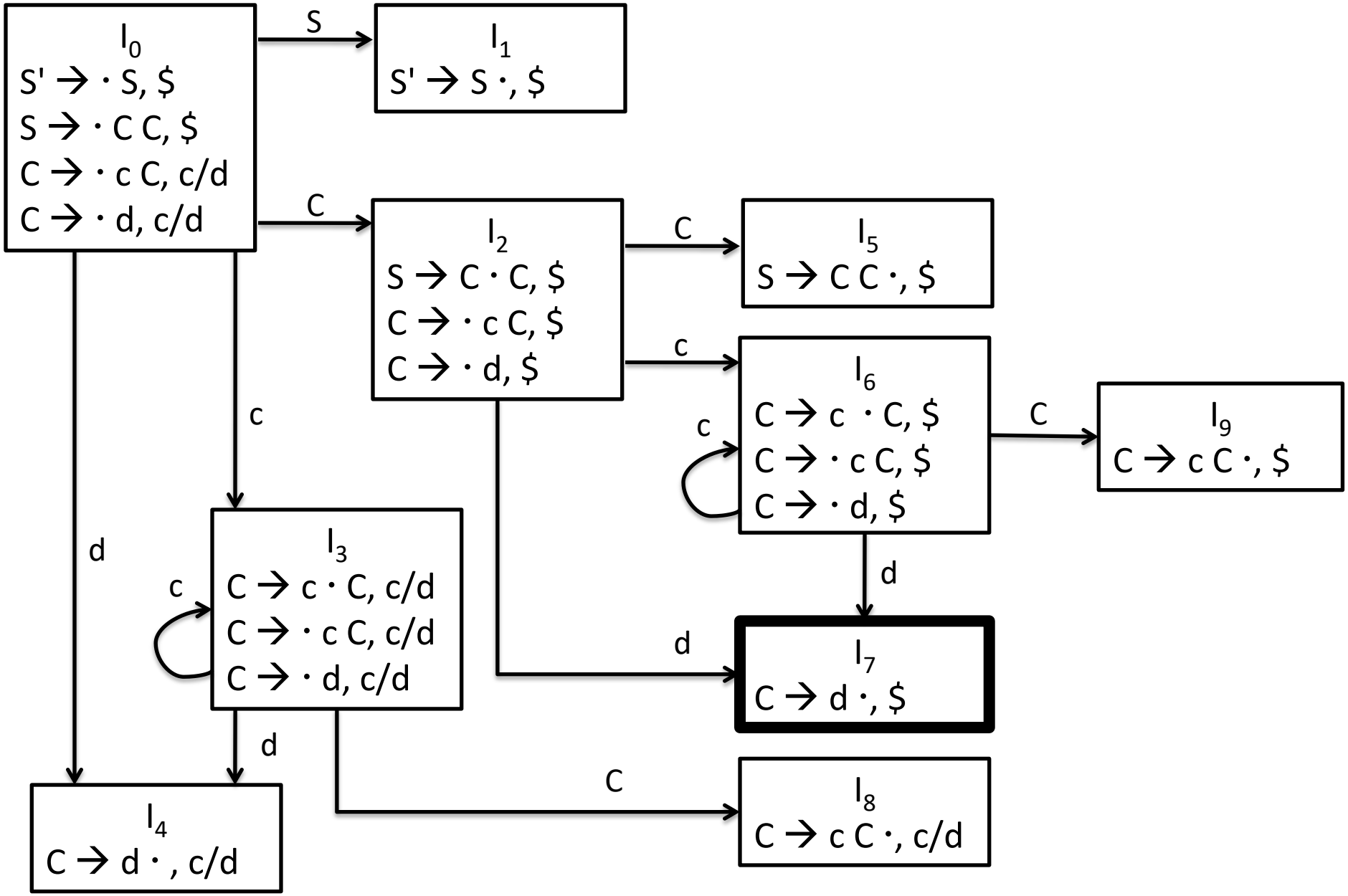

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
d
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
I7
C → d ·, $
d
C
I8
C → c C ·, c/d
I4
C → d · , c/d

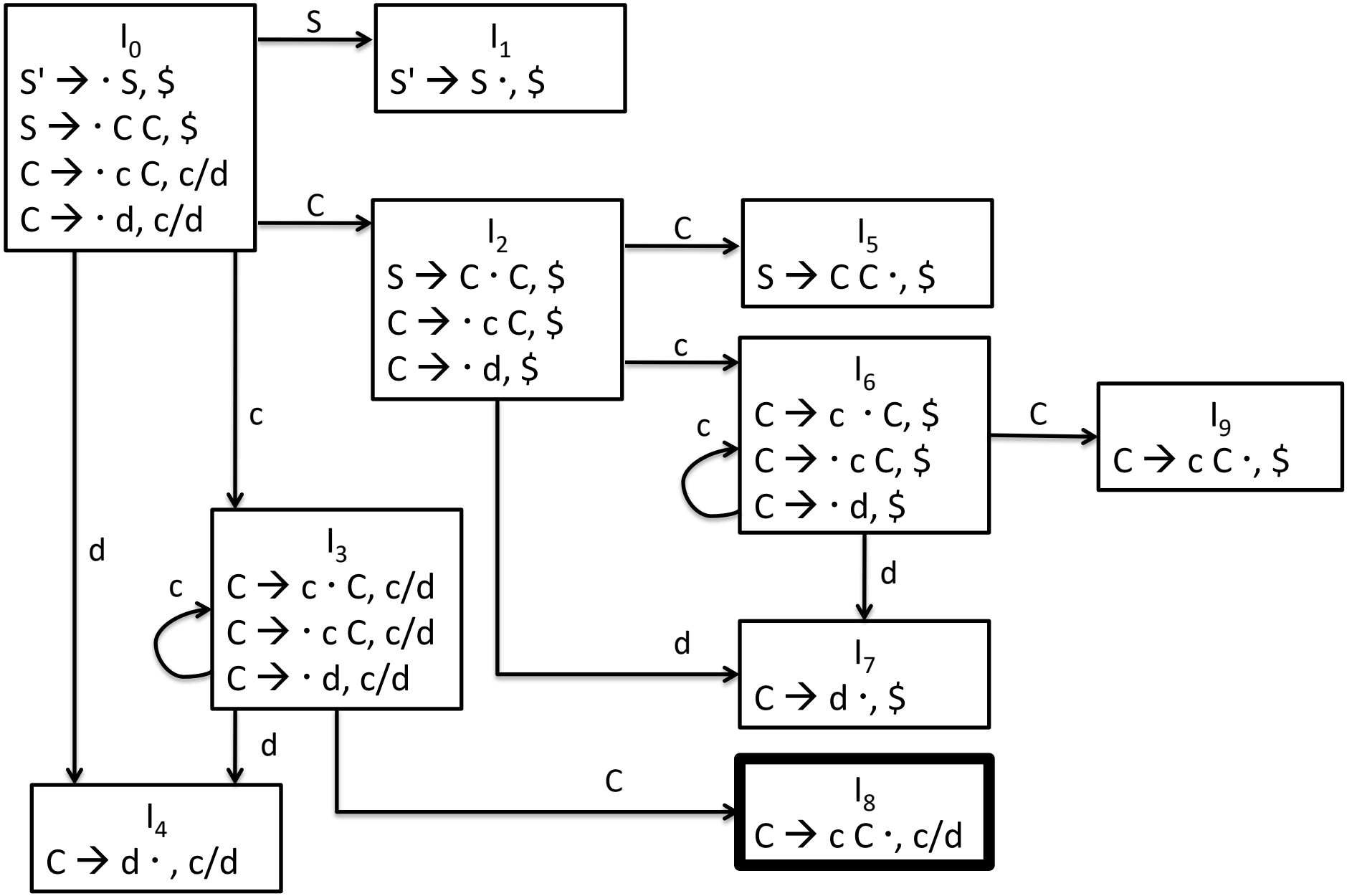

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
d
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
I7
C → d ·, $
d
C
I4
C → d · , c/d
I8
C → c C ·, c/d

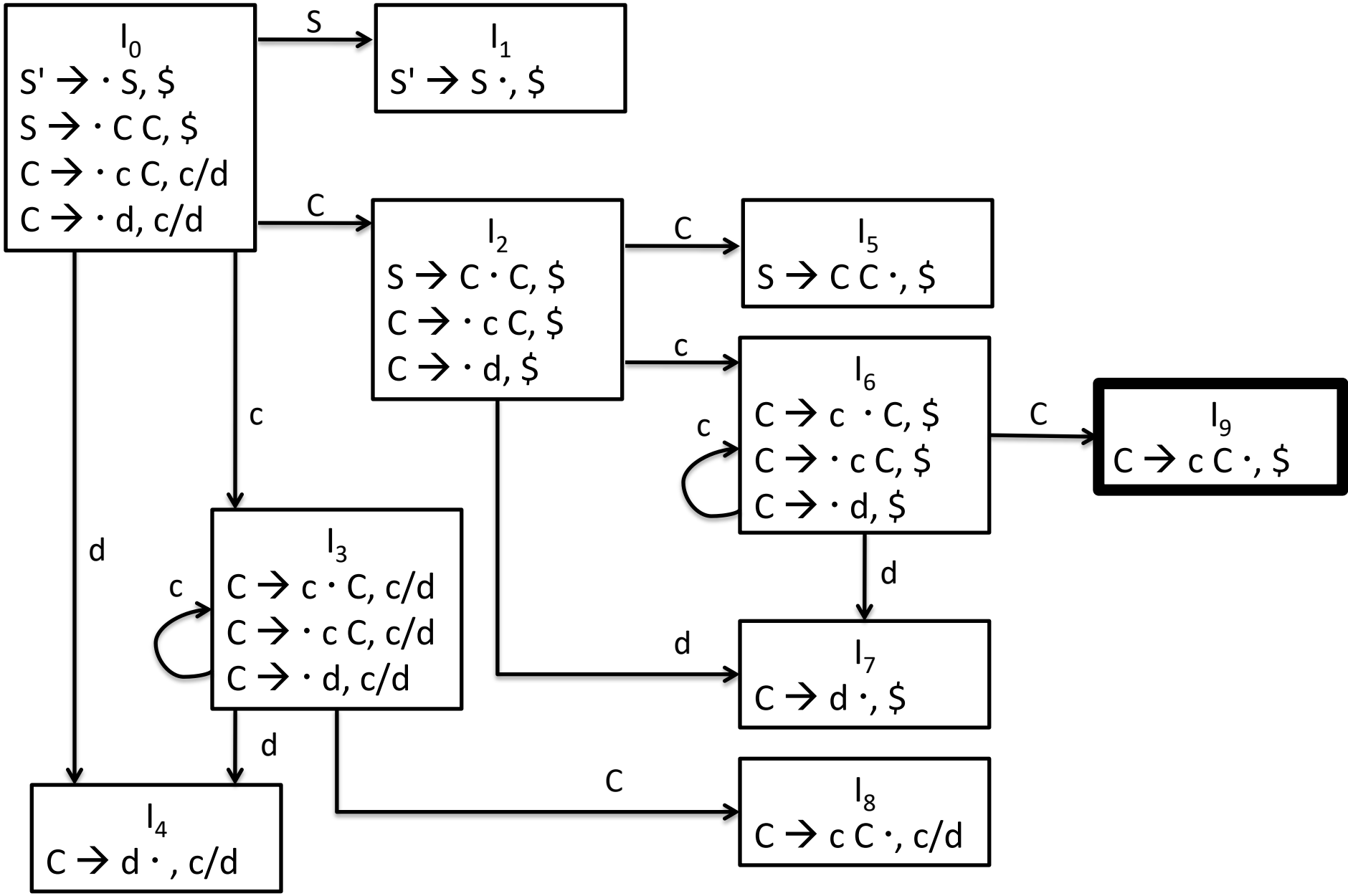

I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
C
I9
C → c C ·, $
d
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
I7
C → d ·, $
d
C
I4
C → d · , c/d
I8
C → c C ·, c/d


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
C
I9
C → c C ·, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
I7
C → d ·, $
d
I4
C → d · , c/d
d
C
I8
C → c C ·, c/d


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
C
I9
C → c C ·, $
d
d
I7
C → d ·, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
d
I4
C → d · , c/d
C
I8
C → c C ·, c/d


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
C
I9
C → c C ·, $
d
d
I7
C → d ·, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
d
I4
C → d · , c/d
C
I8
C → c C ·, c/d


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
C
I9
C → c C ·, $
d
d
I7
C → d ·, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
d
C
I4
C → d · , c/d
I8
C → c C ·, c/d


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
C
I9
C → c C ·, $
d
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
I7
C → d ·, $
d
d
C
I4
C → d · , c/d
I8
C → c C ·, c/d

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1					
2					
3					
4					
5					
6					
7					
8					
9					

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2					
3					
4					
5					
6					
7					
8					
9					

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3					
4					
5					
6					
7					
8					
9					

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6					
7					
8					
9					

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6					
7					
8					
9					

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6					
7					
8					
9					

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6	s6	s7			9
7					
8					
9					

Canonical LR(1) parsing table

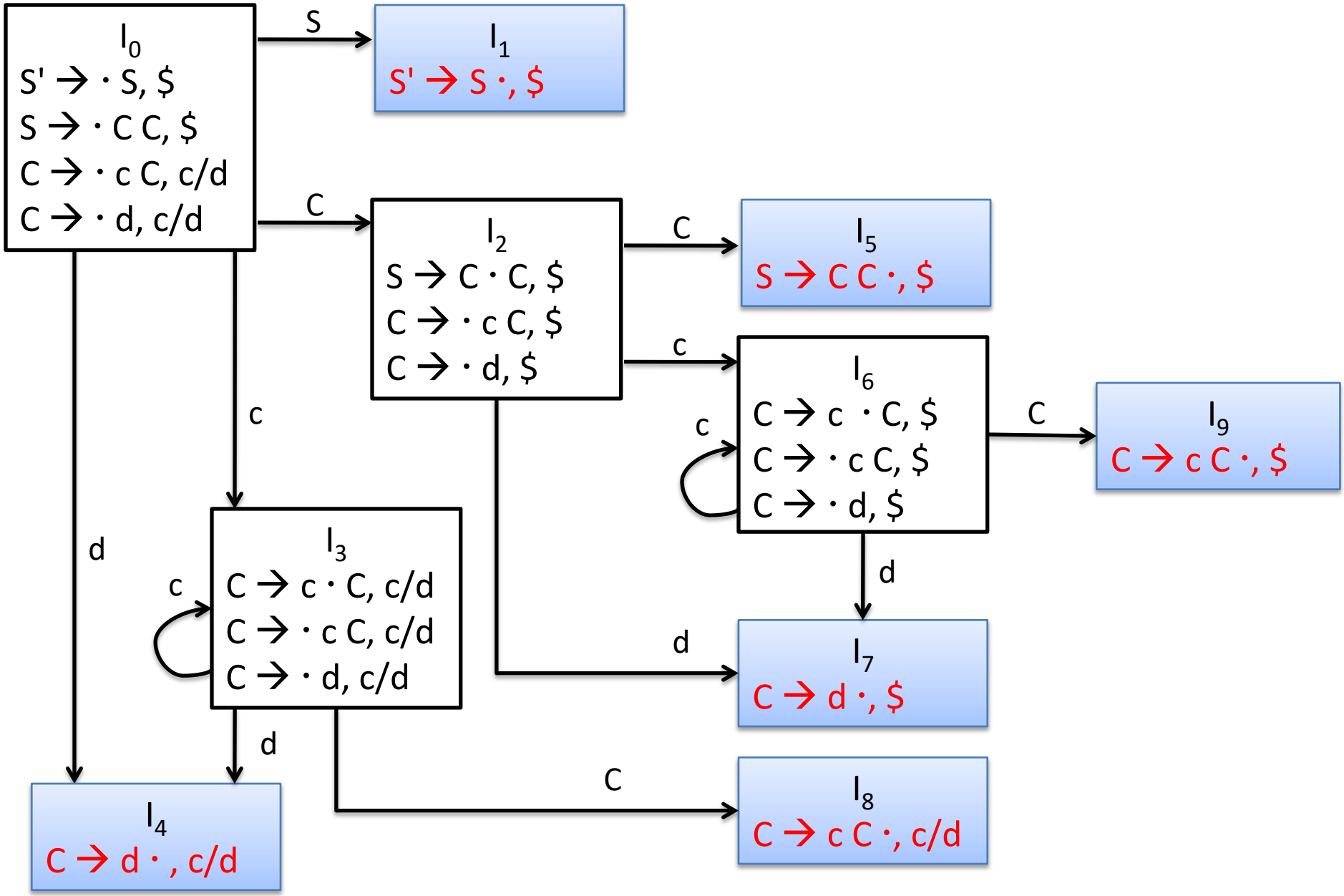
STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6	s6	s7			9
7					
8					
9					

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6	s6	s7			9
7					
8					
9					

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6	s6	s7			9
7					
8					
9					


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
C
I9
C → c C ·, $
d
d
I7
C → d ·, $
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
d
I4
C → d · , c/d
C
I8
C → c C ·, c/d

1. S → C C
2. C → c C
3. C → d

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6	s6	s7			9
7					
8					
9					

I_1
S' → S ·, $

1. S → C C
2. C → c C
3. C → d

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4					
5					
6	s6	s7			9
7					
8					
9					

I_4
C → d ·, c/d

1. S → C C
2. C → c C
3. C → d

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5					
6	s6	s7			9
7					
8					
9					

I_5
S → C C ·, $

1. S → C C
2. C → c C
3. C → d

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	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					
8					
9					

I_7
C → d ·, $

1. S → C C
2. C → c C
3. C → d

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	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					
9					

I_8
C → c C ·, c/d

1. S → C C
2. C → c C
3. C → d

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	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					

I_9
C → c C ·, $

1. S → C C
2. C → c C
3. C → d

Canonical LR(1) parsing table

STATE	ACTION			GOTO	
	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		

Canonical LR(1) parsing table

1. S → C C
2. C → c C
3. C → d

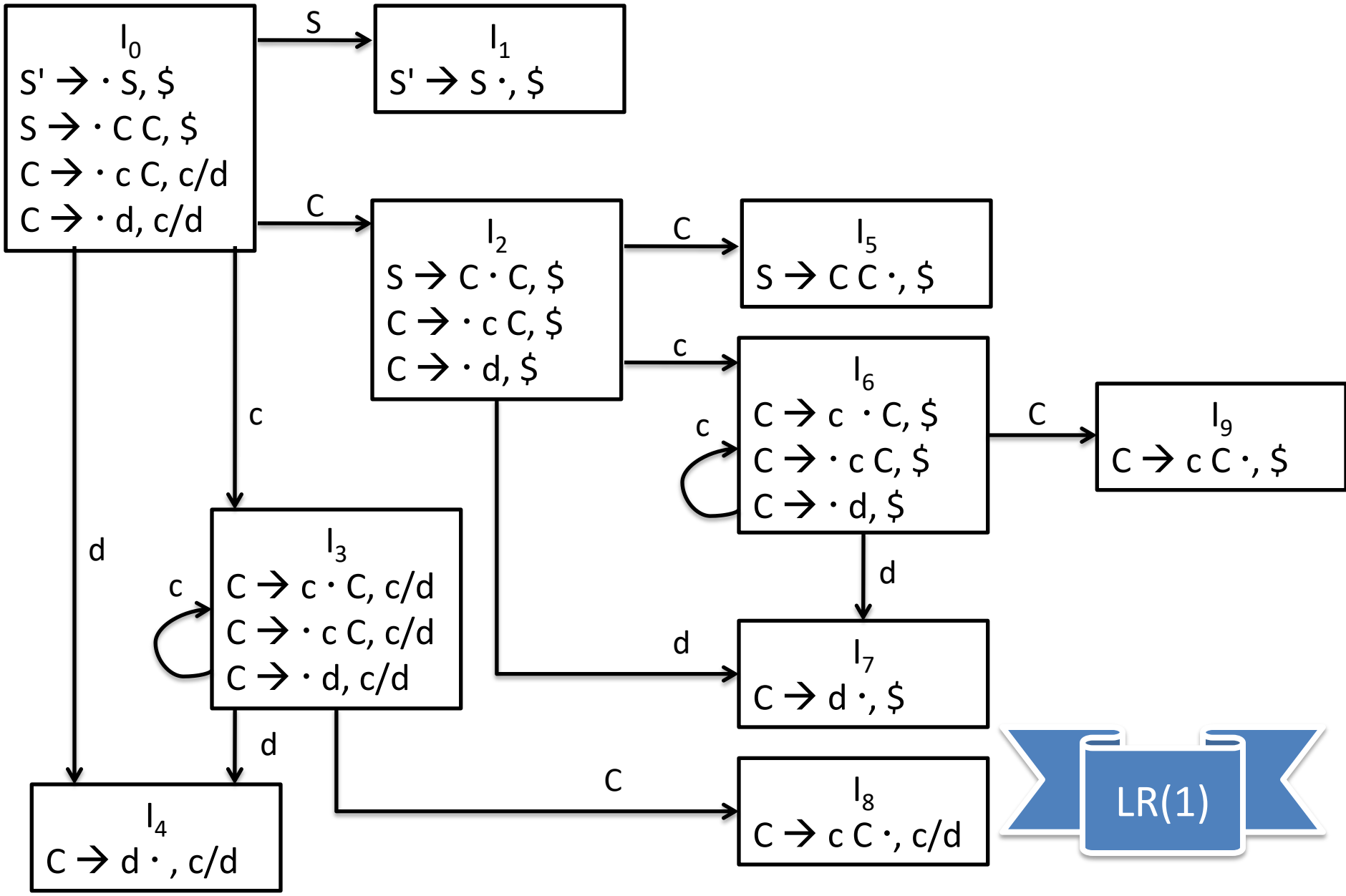
STATE	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

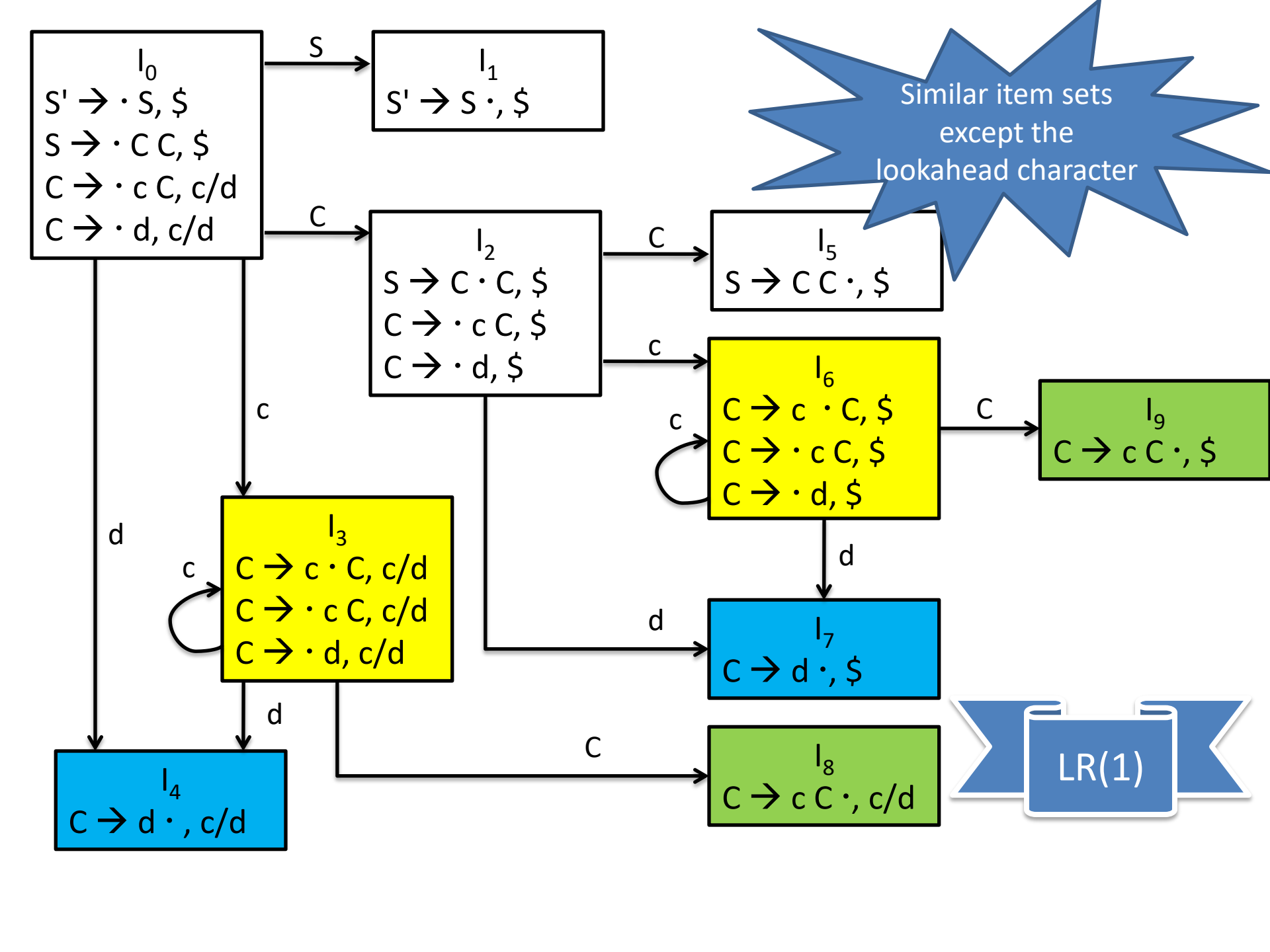
Notes

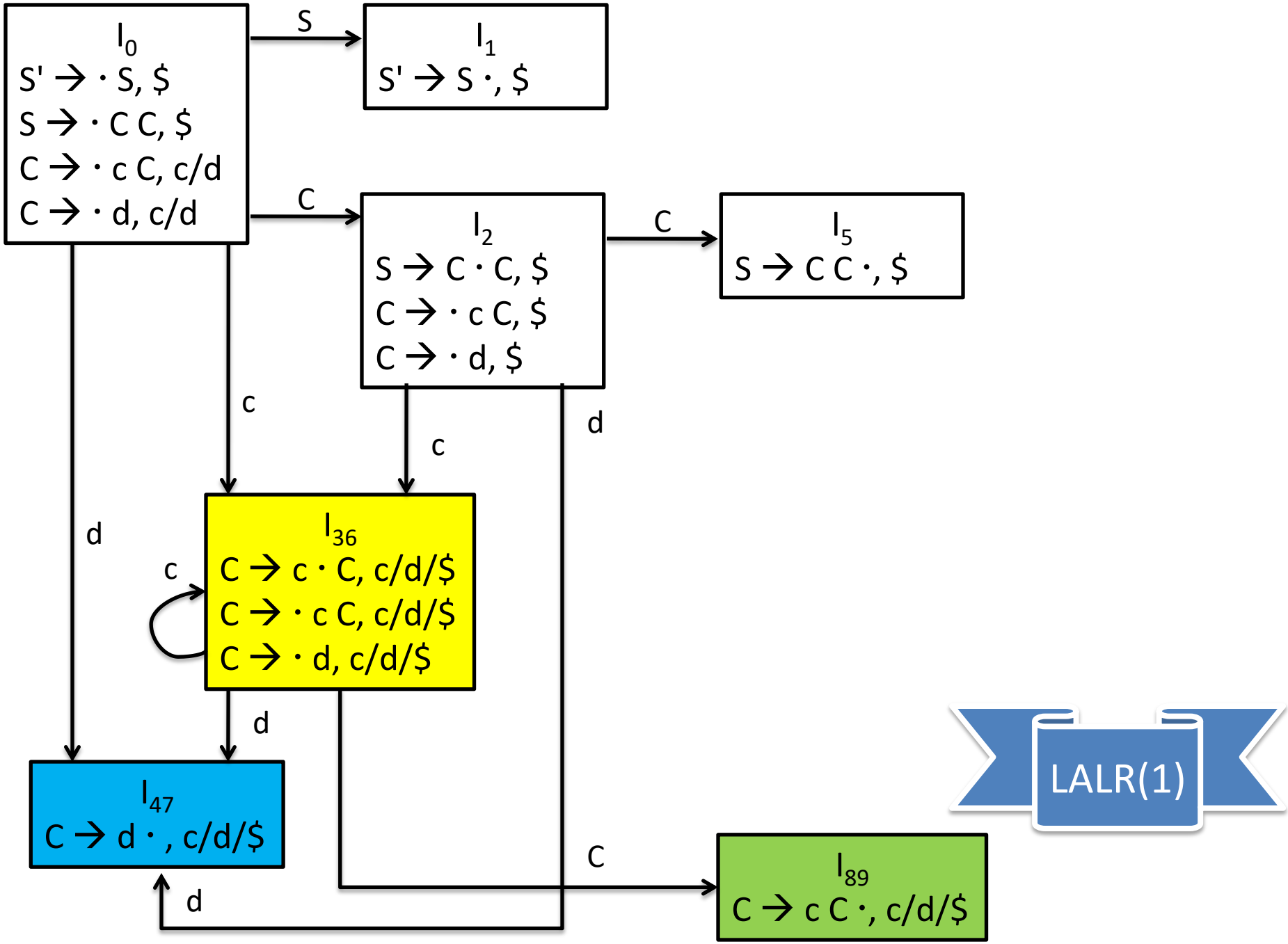
- The table formed from the parsing action and goto functions is called **the canonical LR(1) parsing table**.
- An LR parser using this table is called a **canonical-LR(1) parser**.
- If the parsing action function has no multiply defined entries, then the given grammar is called **an LR(1) grammar**. As before, we omit the "(1)" if it is understood.

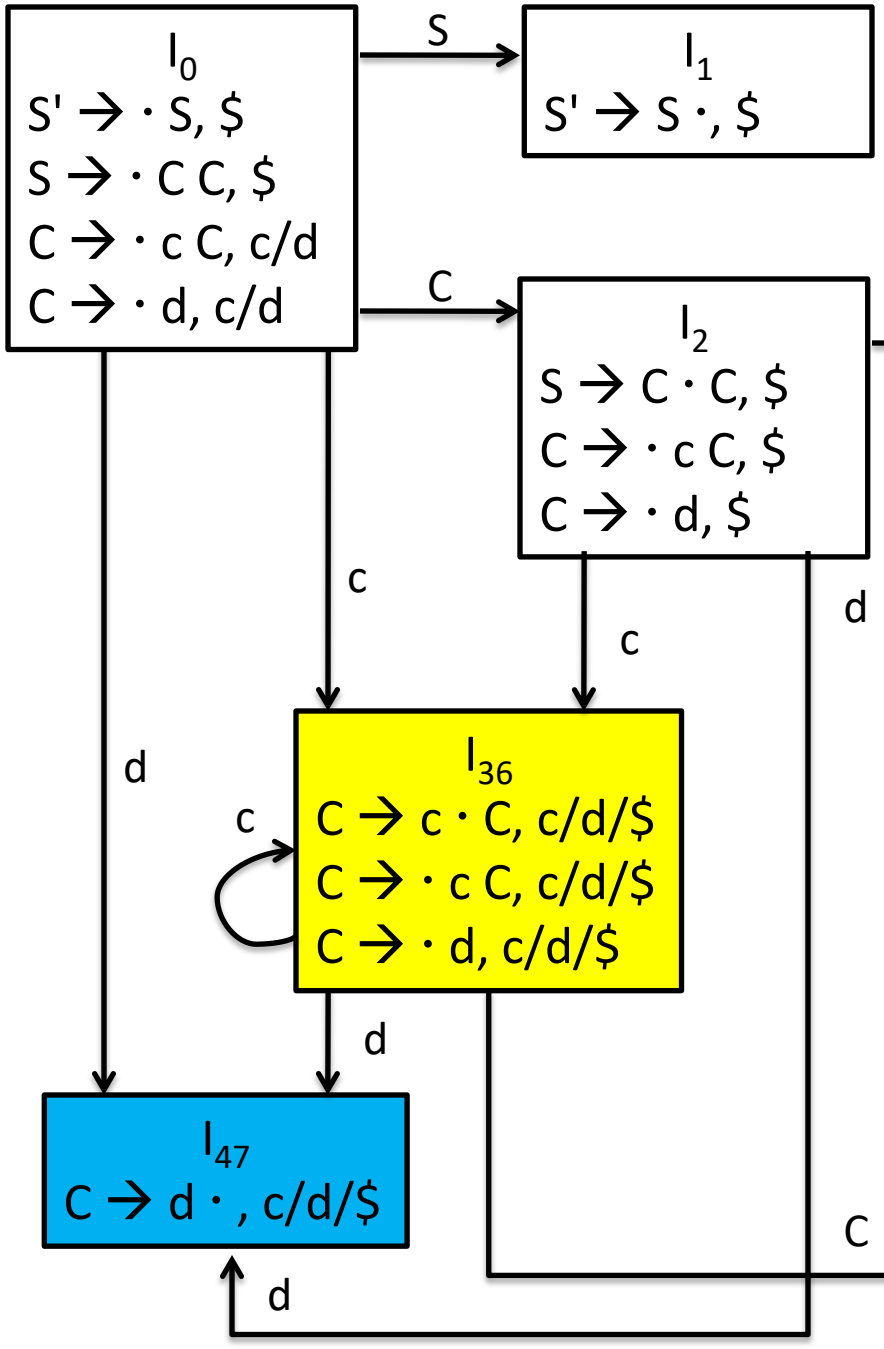
LALR Parsing [Look-Ahead LR]

- This method is often used in practice, because the tables obtained by it are **considerably smaller** than the canonical LR tables, yet most common syntactic constructs of programming languages can be expressed conveniently by an LALR grammar.
- It is **much easier and more economical** to construct SLR and LALR tables than the canonical LR tables.


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
C
I9
C → c C ·, $
d
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
c
d
I7
C → d ·, $
d
d
I4
C → d · , c/d
C
I8
C → c C ·, c/d
LR(1)


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
I6
C → c · C, $
C → · c C, $
C → · d, $
c
C
I9
C → c C ·, $
d
I7
C → d ·, $
d
c
I3
C → c · C, c/d
C → · c C, c/d
C → · d, c/d
d
d
I4
C → d · , c/d
C
I8
C → c C ·, c/d
Similar item sets except the lookahead character
LR(1)


I0
S' → · S, $
S → · C C, $
C → · c C, c/d
C → · d, c/d
S
I1
S' → S ·, $
C
I2
S → C · C, $
C → · c C, $
C → · d, $
C
I5
S → C C ·, $
c
d
c
d
I36
C → c · C, c/d/$
C → · c C, c/d/$
C → · d, c/d/$
c
d
I47
C → d · , c/d/$
d
C
I89
C → c C ·, c/d/$
LALR(1)



STATE	ACTION			GOTO	
	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		

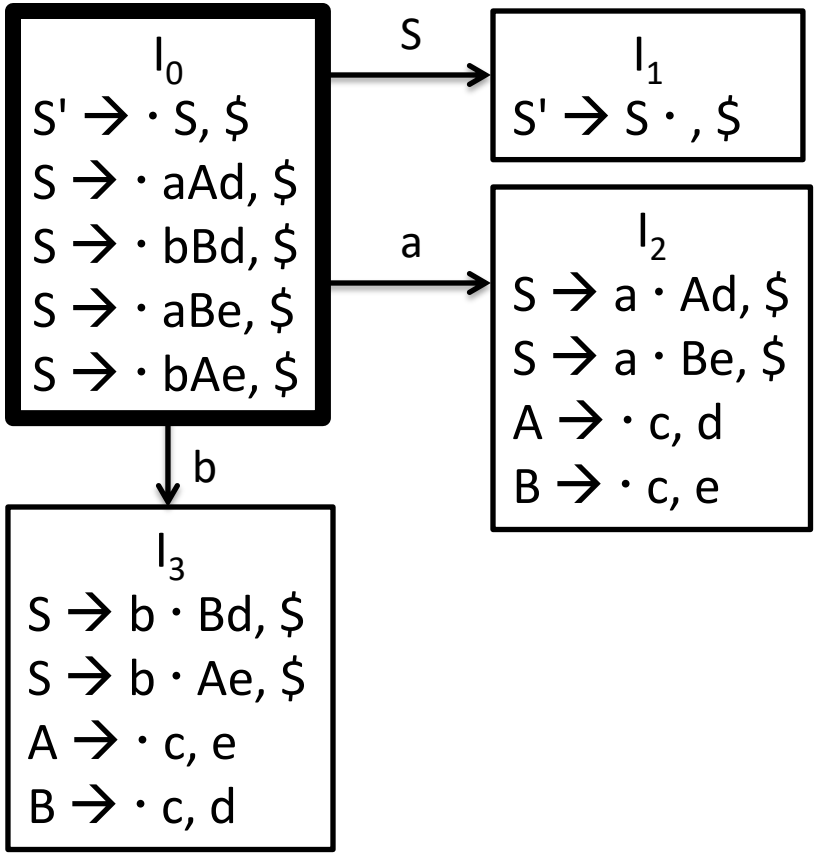
Issue in merger

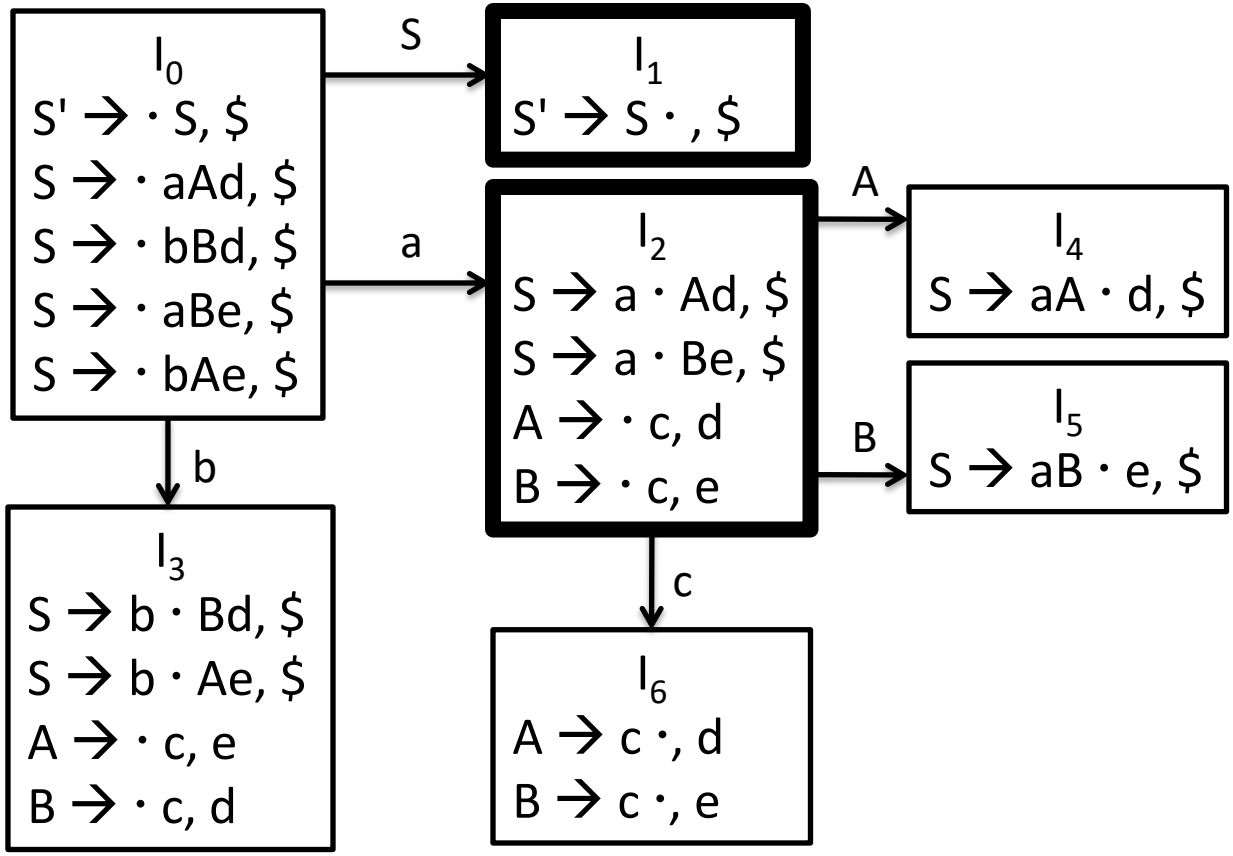
- The merging of states with common cores can never produce a shift/reduce conflict that was not present in one of the original states, because shift actions depend only on the core, not the lookahead.
- It is possible, however, that a merger will produce a reduce/reduce conflict.

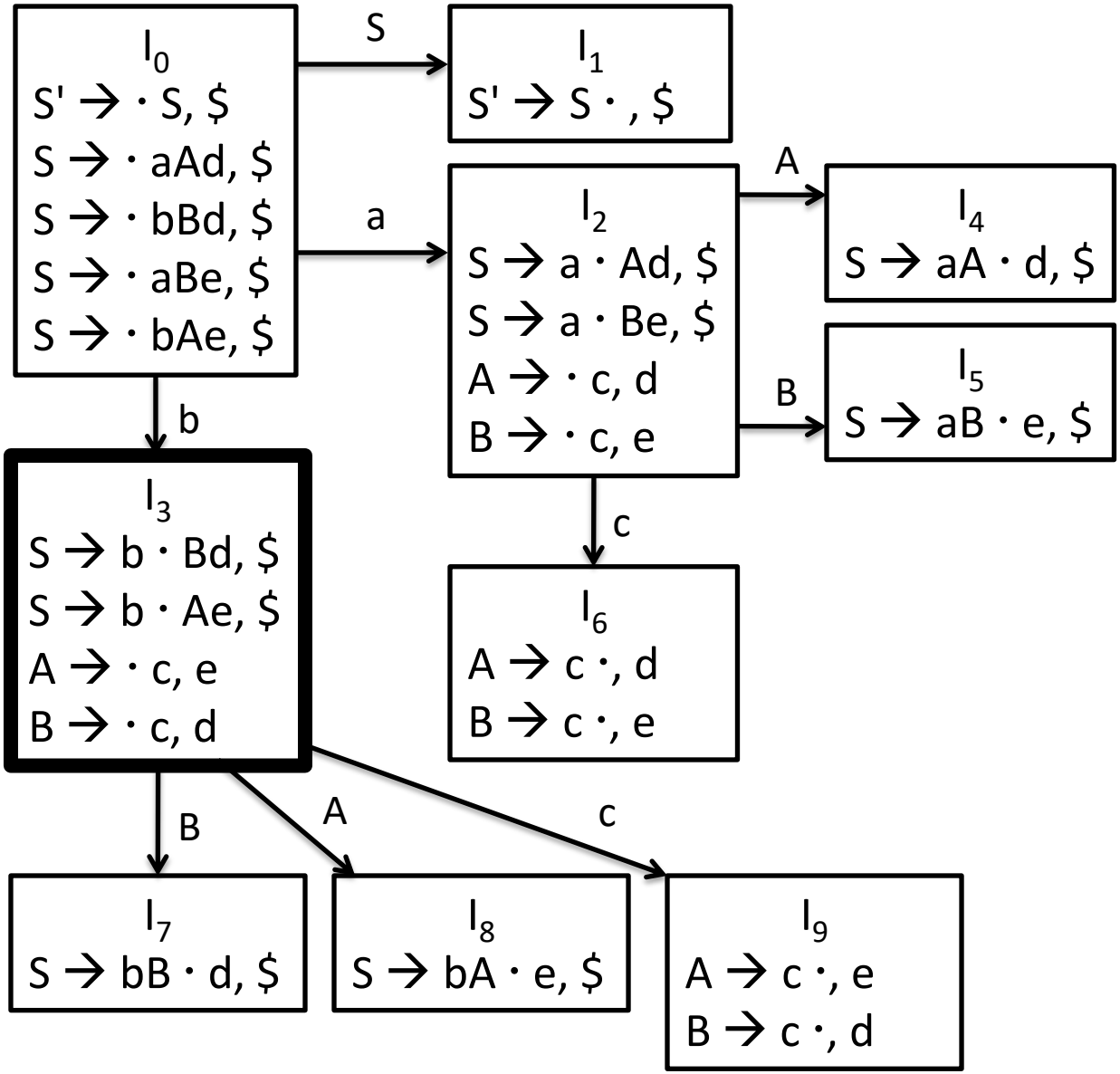
Example to demonstrate reduce/reduce conflict after merger

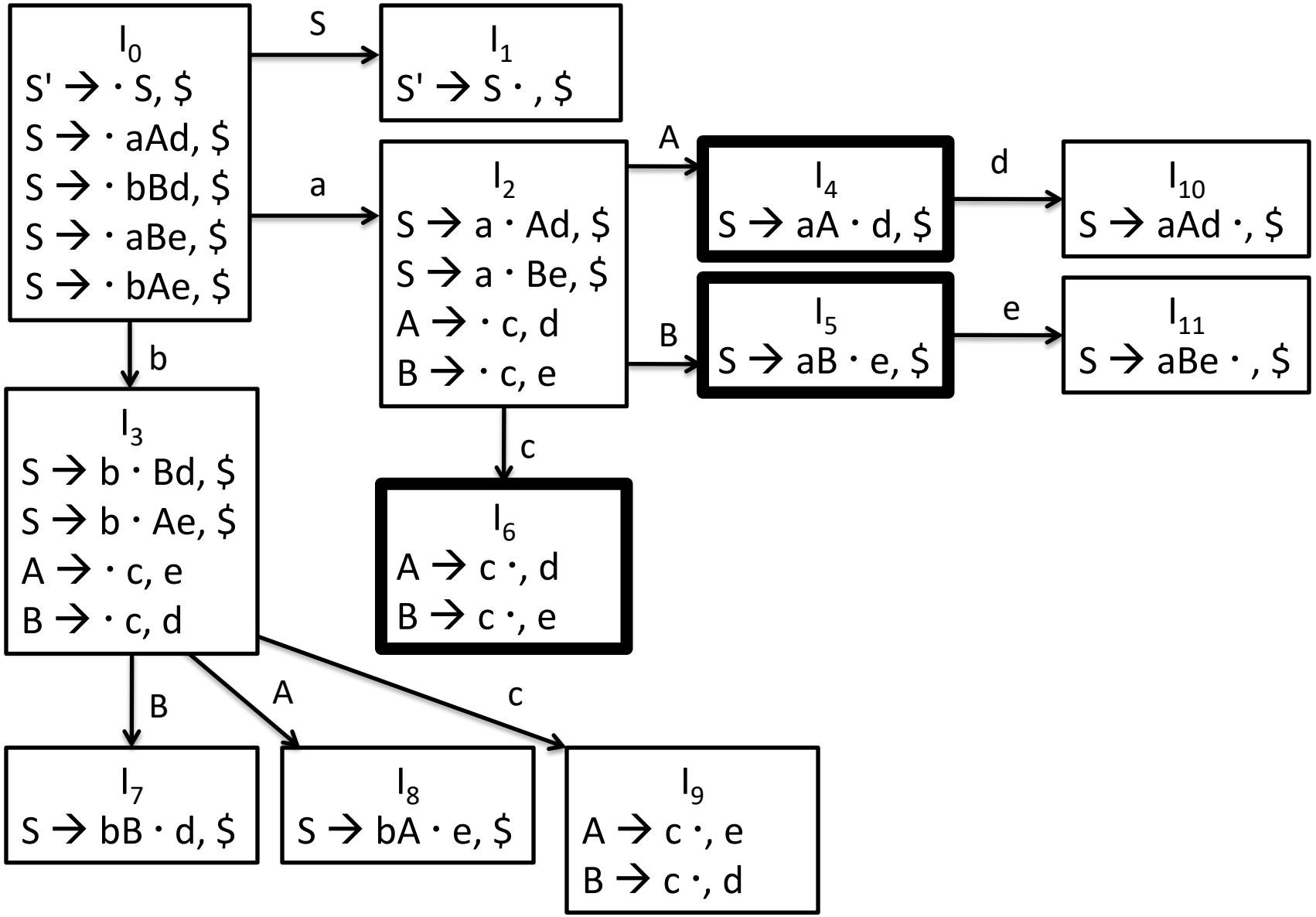
- S' → S
  S → aAd | bBd | aBe | bAe
  A → c
  B → c

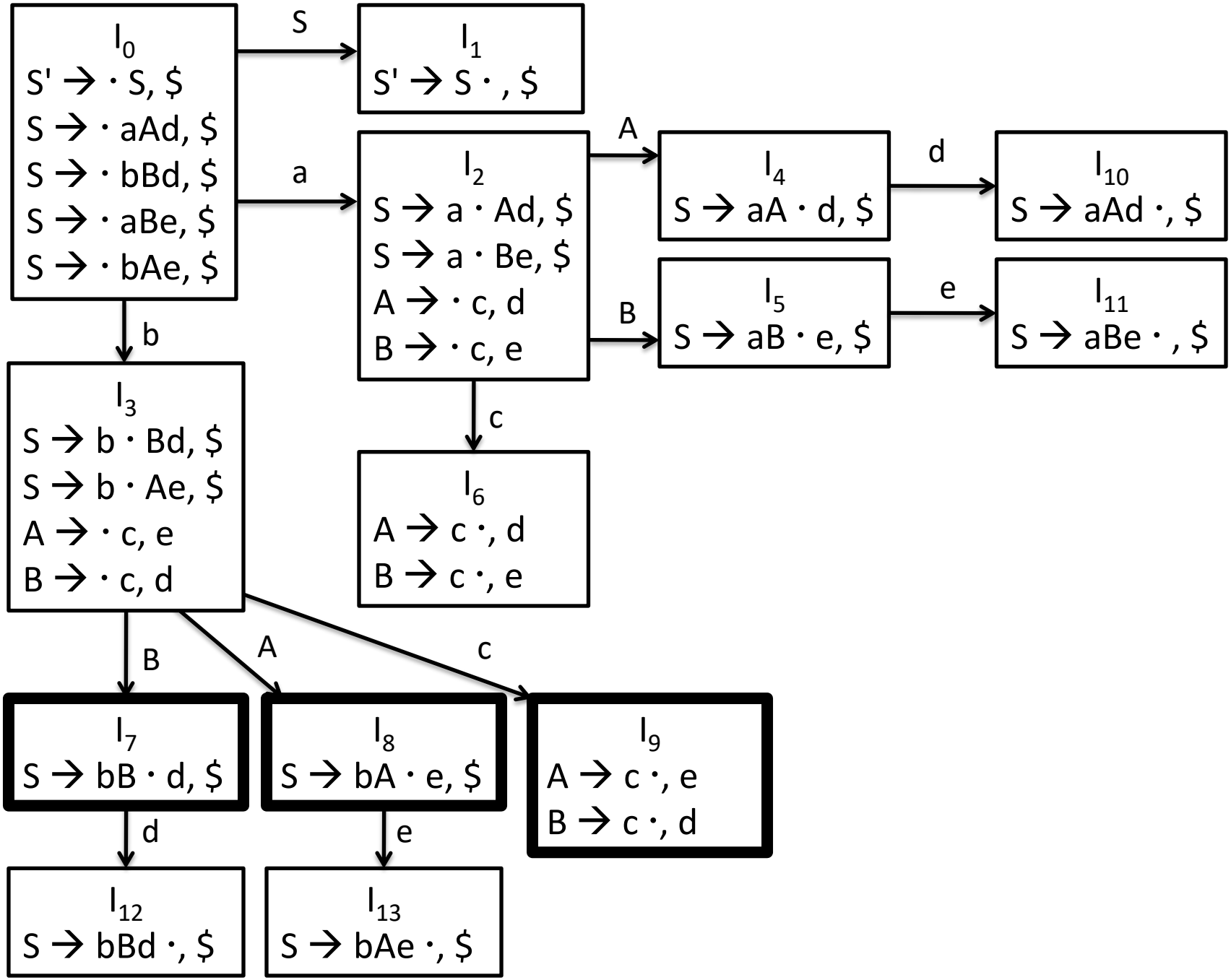
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $

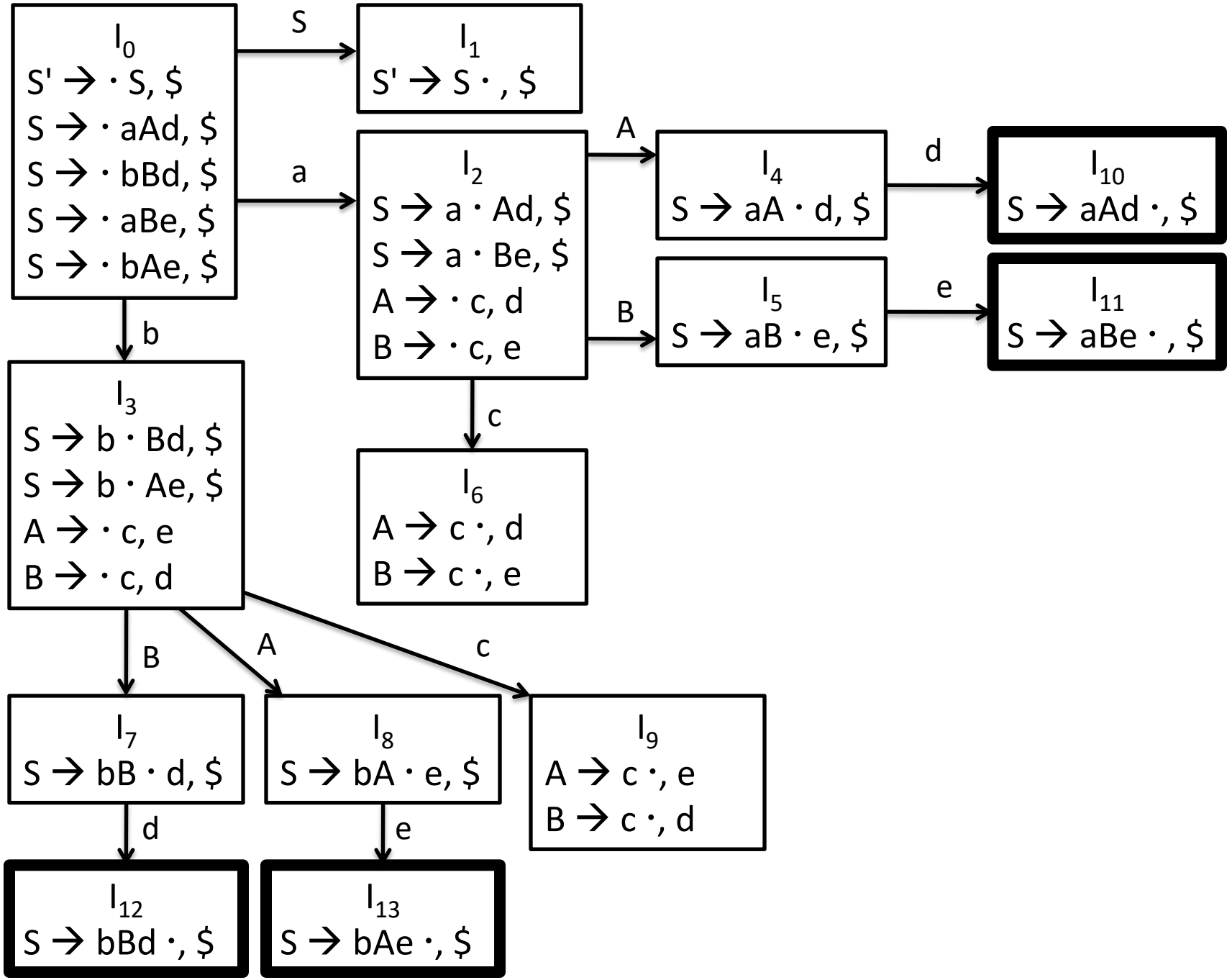

I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d

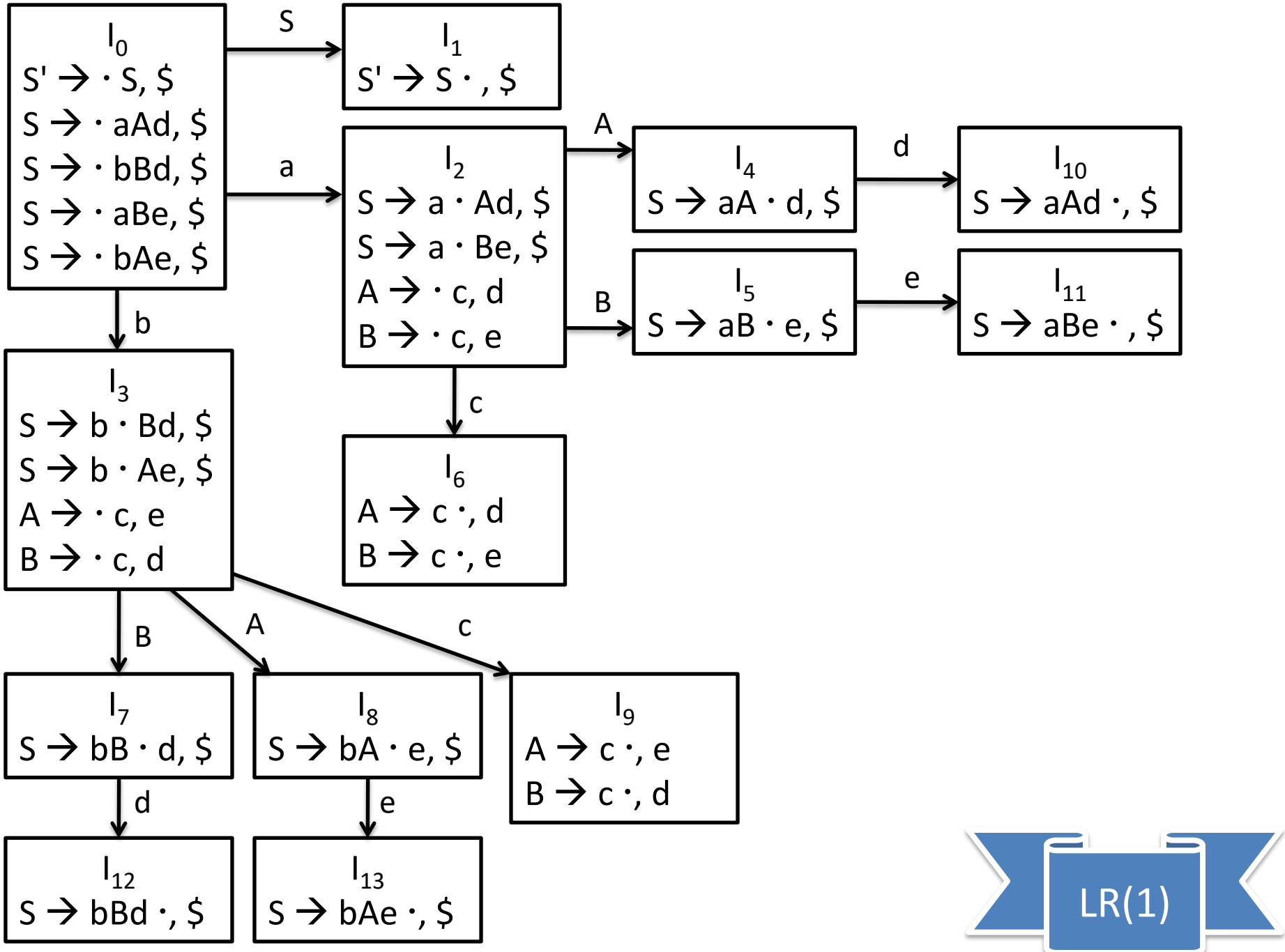

I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
A
I4
S → aA · d, $
B
I5
S → aB · e, $
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d
c
I6
A → c ·, d
B → c ·, e

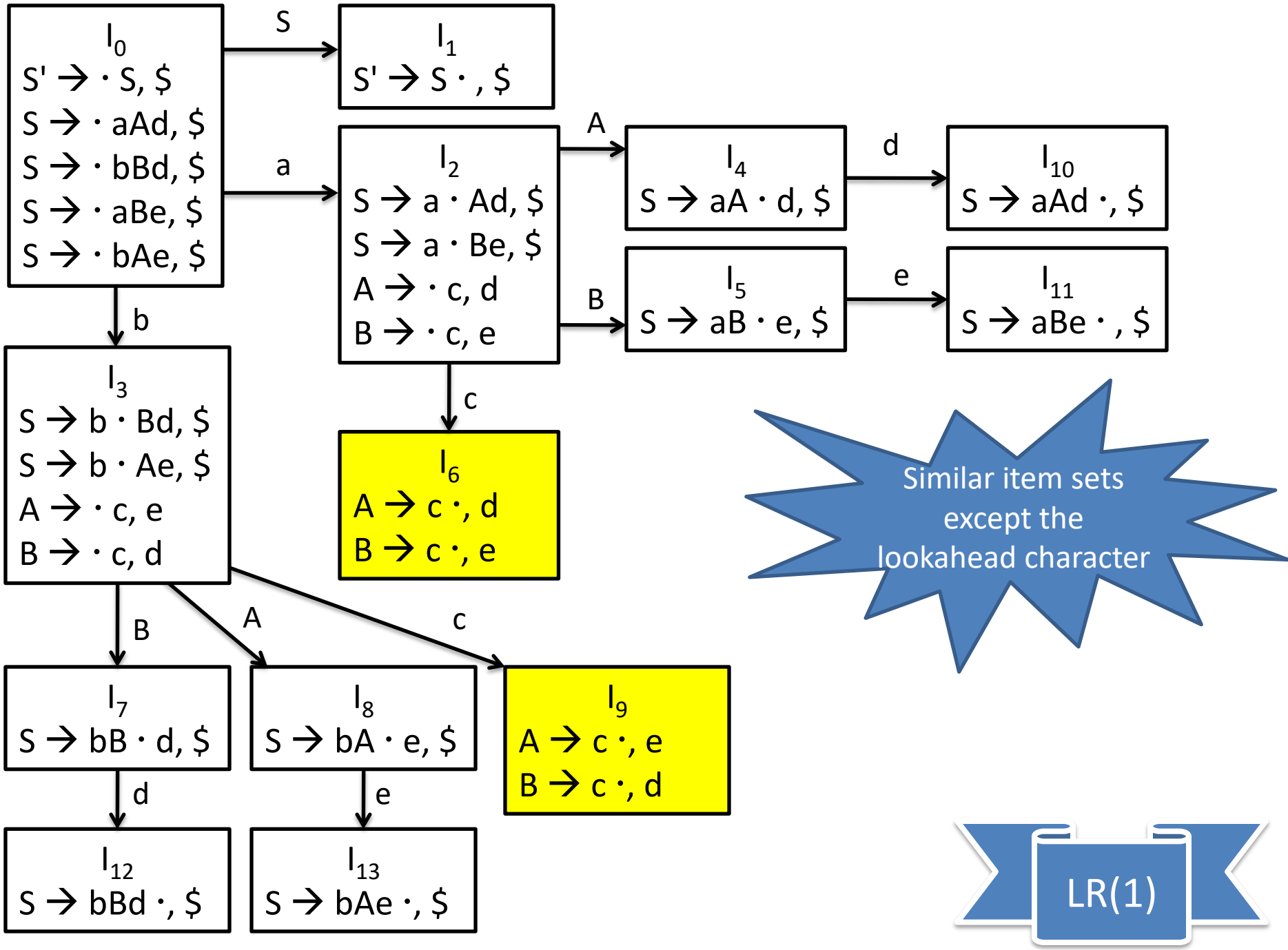

I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
A
I4
S → aA · d, $
B
I5
S → aB · e, $
c
I6
A → c ·, d
B → c ·, e
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d
B
I7
S → bB · d, $
A
I8
S → bA · e, $
c
I9
A → c ·, e
B → c ·, d

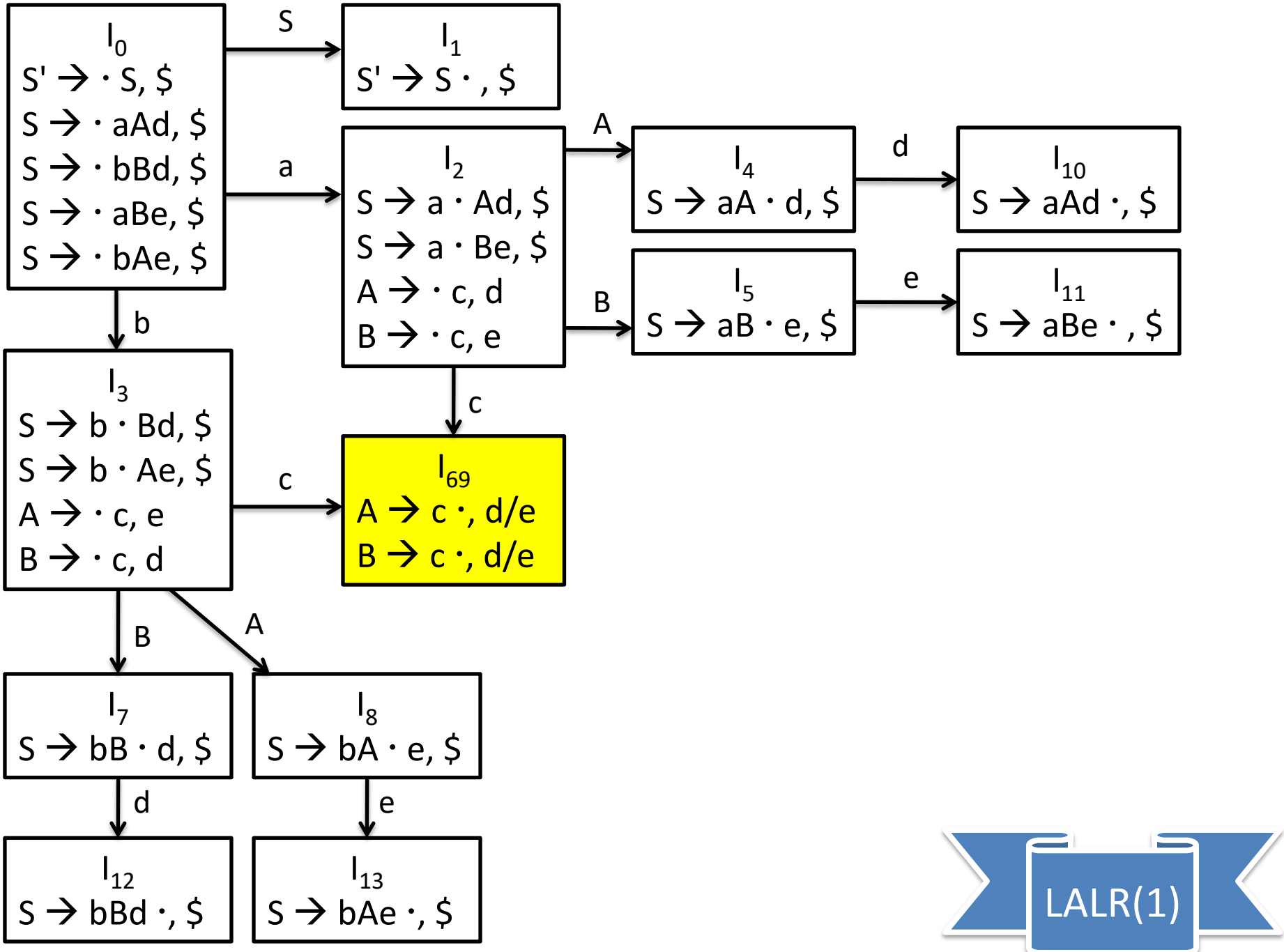

I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
A
I4
S → aA · d, $
d
I10
S → aAd ·, $
B
I5
S → aB · e, $
e
I11
S → aBe · , $
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d
c
I6
A → c ·, d
B → c ·, e
B
I7
S → bB · d, $
A
I8
S → bA · e, $
c
I9
A → c ·, e
B → c ·, d

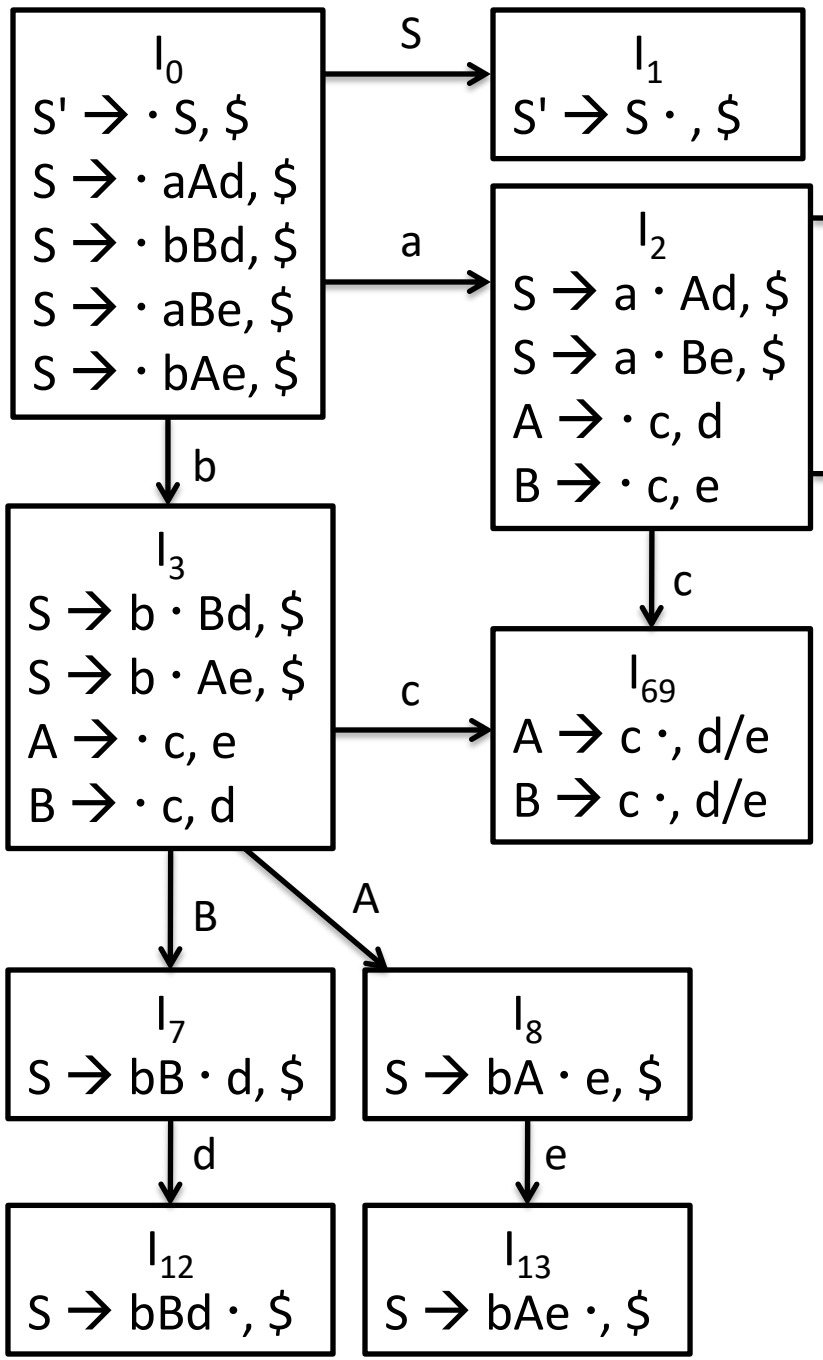

I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
A
I4
S → aA · d, $
d
I10
S → aAd ·, $
B
I5
S → aB · e, $
e
I11
S → aBe · , $
c
I6
A → c ·, d
B → c ·, e
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d
B
I7
S → bB · d, $
A
I8
S → bA · e, $
c
I9
A → c ·, e
B → c ·, d
d
I12
S → bBd ·, $
e
I13
S → bAe ·, $


I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
A
I4
S → aA · d, $
d
I10
S → aAd ·, $
B
I5
S → aB · e, $
e
I11
S → aBe · , $
c
I6
A → c ·, d
B → c ·, e
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d
B
I7
S → bB · d, $
d
I12
S → bBd ·, $
A
I8
S → bA · e, $
e
I13
S → bAe ·, $
c
I9
A → c ·, e
B → c ·, d


I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
A
I4
S → aA · d, $
d
I10
S → aAd ·, $
B
I5
S → aB · e, $
e
I11
S → aBe · , $
c
I6
A → c ·, d
B → c ·, e
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d
B
I7
S → bB · d, $
d
I12
S → bBd ·, $
A
I8
S → bA · e, $
e
I13
S → bAe ·, $
c
I9
A → c ·, e
B → c ·, d
LR(1)

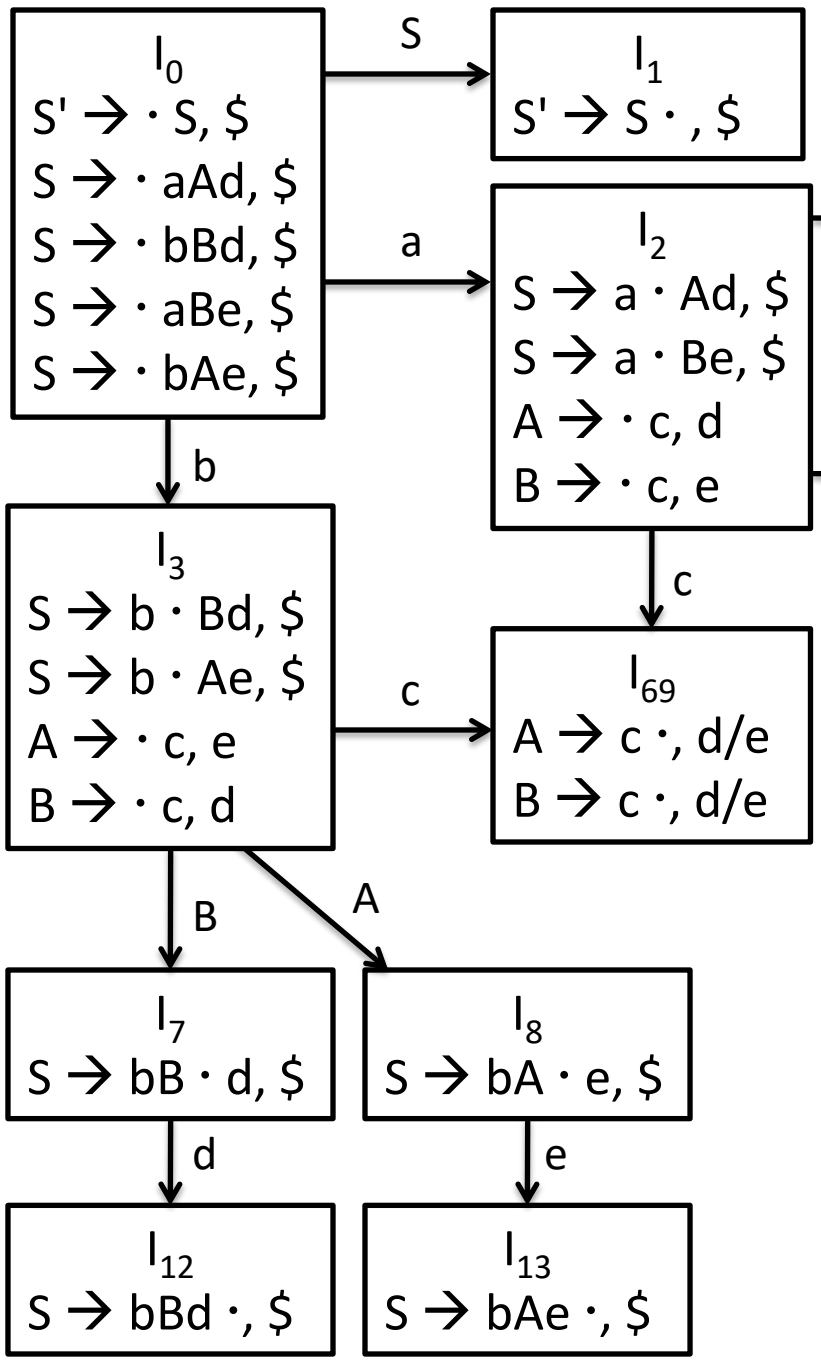

I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
A
I4
S → aA · d, $
d
I10
S → aAd ·, $
B
I5
S → aB · e, $
e
I11
S → aBe · , $
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d
c
I6
A → c ·, d
B → c ·, e
Similar item sets except the lookahead character
B
A
c
I7
S → bB · d, $
I8
S → bA · e, $
I9
A → c ·, e
B → c ·, d
d
e
I12
S → bBd ·, $
I13
S → bAe ·, $
LR(1)


I0
S' → · S, $
S → · aAd, $
S → · bBd, $
S → · aBe, $
S → · bAe, $
S
I1
S' → S · , $
a
I2
S → a · Ad, $
S → a · Be, $
A → · c, d
B → · c, e
A
I4
S → aA · d, $
d
I10
S → aAd ·, $
B
I5
S → aB · e, $
e
I11
S → aBe · , $
b
I3
S → b · Bd, $
S → b · Ae, $
A → · c, e
B → · c, d
c
I69
A → c ·, d/e
B → c ·, d/e
B
A
I7
S → bB · d, $
I8
S → bA · e, $
d
e
I12
S → bBd ·, $
I13
S → bAe ·, $
LALR(1)



1. S → aAd
2. S → bBd
3. S → aBe
4. S → bAe
5. A → c
6. B → c

STATE	ACTION						GOTO		
	a	b	c	d	e	$	S	A	B
0	s2	s3					1		
1						acc			
2			s69					4	5
3			s69					8	7
4				s10					
5					s11				
69				r5/r6	r5/r6				
7				s12					
8					s13				
10						r1			
11						r3			
12						r2			
13						r4			



1. S → aAd
2. S → bBd
3. S → aBe
4. S → bAe
5. A → c
6. B → c

reduce/reduce conflict in LALR(1) parsing table

STATE	ACT								
	a	b	c						
0	s2	s3							
1						cc			
2			s69					4	5
3			s69					8	7
4				s10					
5					s11				
69				r5/r6	r5/r6				
7				s12					
8					s13				
10						r1			
11						r3			
12						r2			
13						r4			

Show that the grammar is LALR(1) but not SLR(1)

S → A a | b A c | d c | b d a

A → d

DIY

Show that the grammar is LR(1) but not LALR(1)

S → A a | b A c | B c | b B a

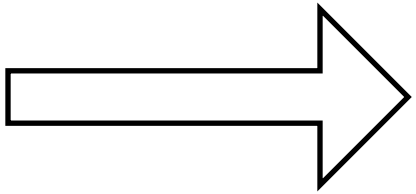
A → d

B → d

DIY

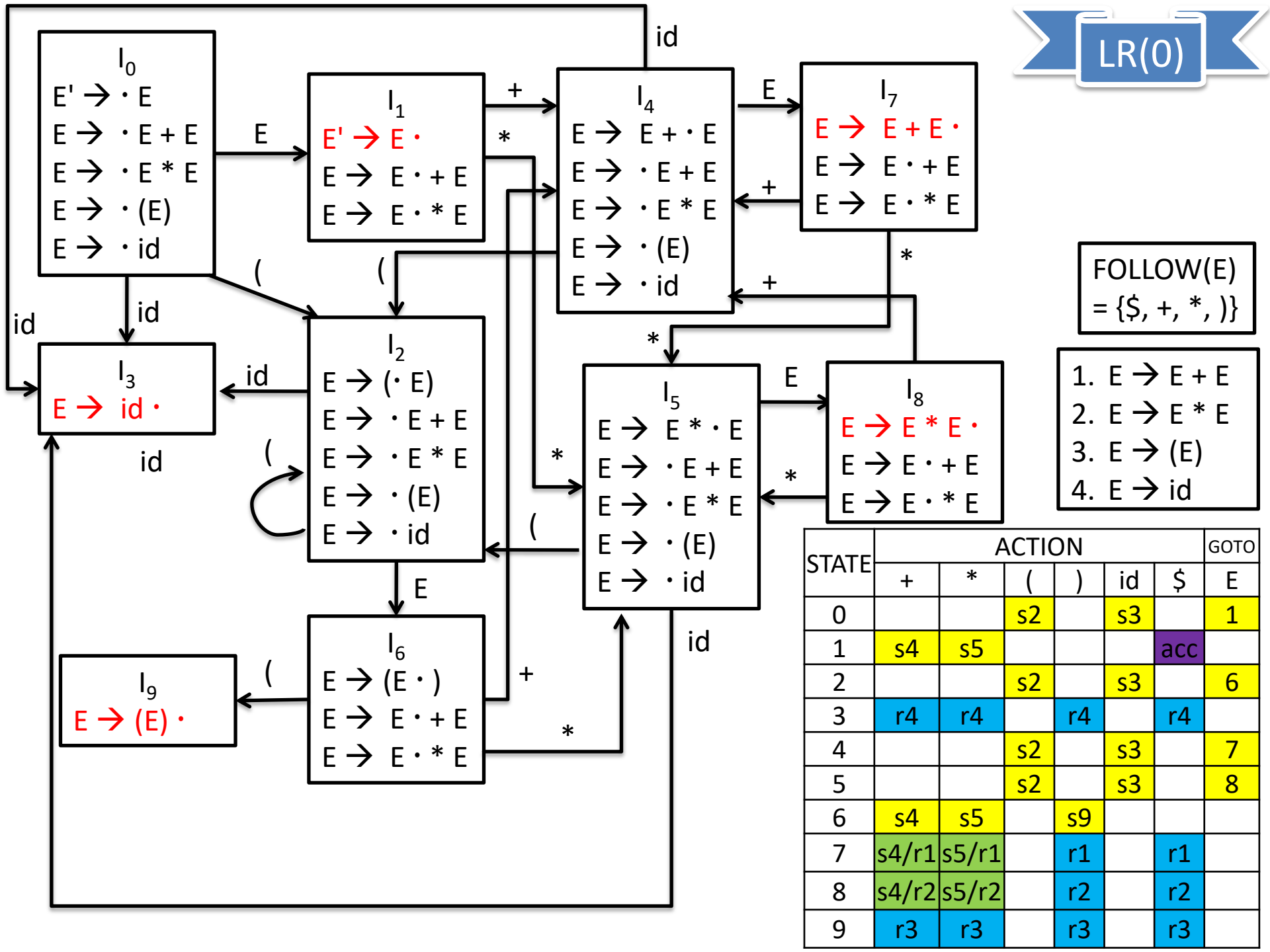
Ambiguous Grammar

1. E → E + E
2. E → E * E
3. E → (E)
4. E → id



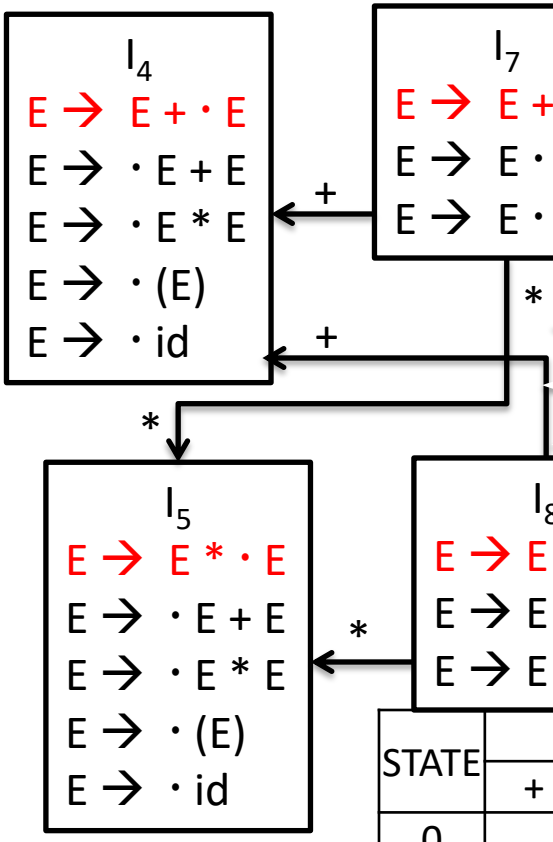
E' → · E
E → · E + E
E → · E * E
E → · (E)
E → · id

- Generate LR(0) Automaton
- Generate LR Parsing table



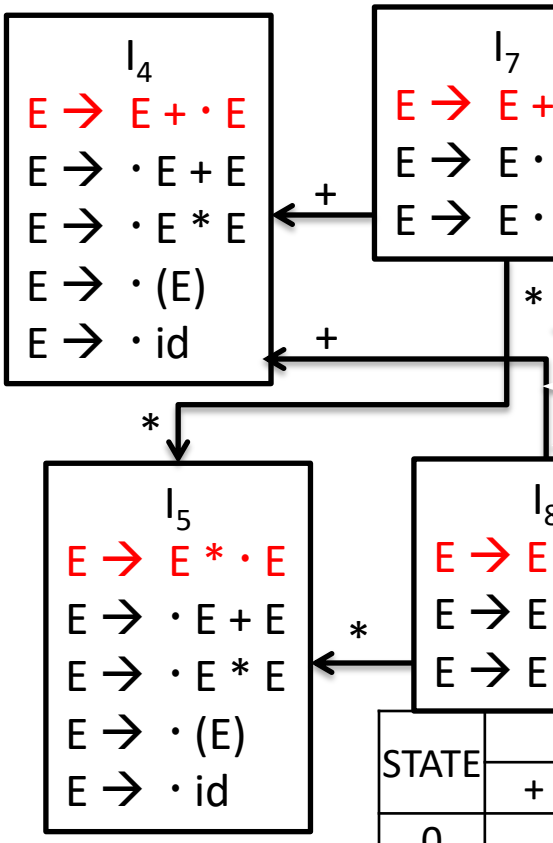
STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0			s2		s3		1
1	s4	s5				acc	
2			s2		s3		6
3	r4	r4		r4		r4	
4			s2		s3		7
5			s2		s3		8
6	s4	s5		s9			
7	s4/r1	s5/r1		r1		r1	
8	s4/r2	s5/r2		r2		r2	
9	r3	r3		r3		r3	

- [7, +]: + operation is performed and again the + operation to be performed.
- [7, *]: + operation is performed and then the * operation to be performed.
- [8, +]: * operation is performed and then the + operation is performed.
- [8, *]: * operation is performed and then the * operation is performed.



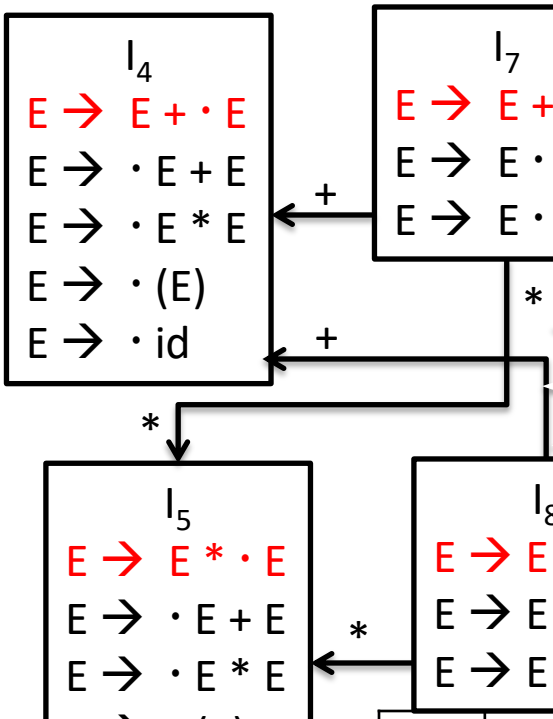
STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0			s2		s3		1
1	s4	s5				acc	
2			s2		s3		6
3	r4	r4		r4		r4	
4			s2		s3		7
5			s2		s3		8
6	s4	s5		s9			
7	s4/r1	s5/r1		r1		r1	
8	s4/r2	s5/r2		r2		r2	
9	r3	r3		r3		r3	

- [7, +]: + operation is performed and again the + operation to be performed. [reduce]
- [7, *]: + operation is performed and then the * operation to be performed.
- [8, +]: * operation is performed and then the + operation is performed.
- [8, *]: * operation is performed and then the * operation is performed. [reduce]



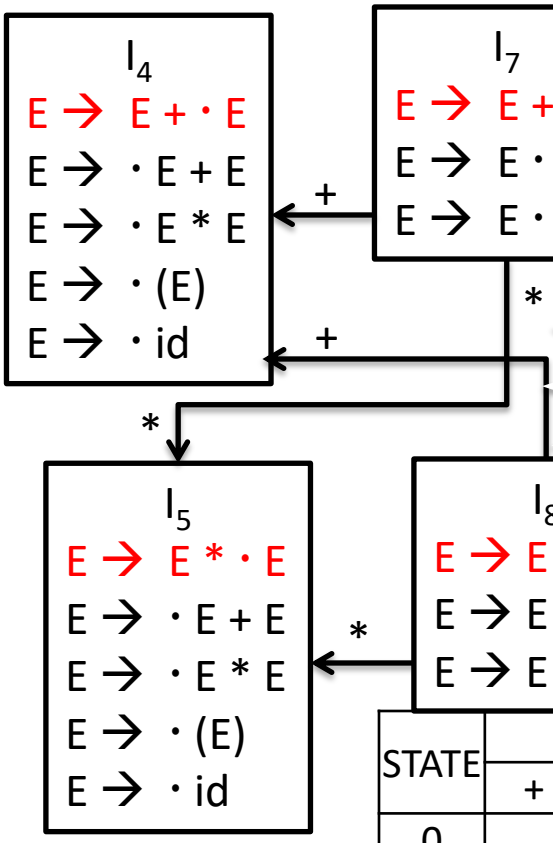
STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0			s2		s3		1
1	s4	s5				acc	
2			s2		s3		6
3	r4	r4		r4		r4	
4			s2		s3		7
5			s2		s3		8
6	s4	s5		s9			
7	r1	s5/r1		r1		r1	
8	s4/r2	r2		r2		r2	
9	r3	r3		r3		r3	

- [7, +]: + operation is performed and again the + operation to be performed. [reduce]
- [7, *]: + operation is performed and then the * operation to be performed.
- [8, +]: * operation is performed and then the + operation is performed. [reduce]
- [8, *]: * operation is performed and then the * operation is performed. [reduce]

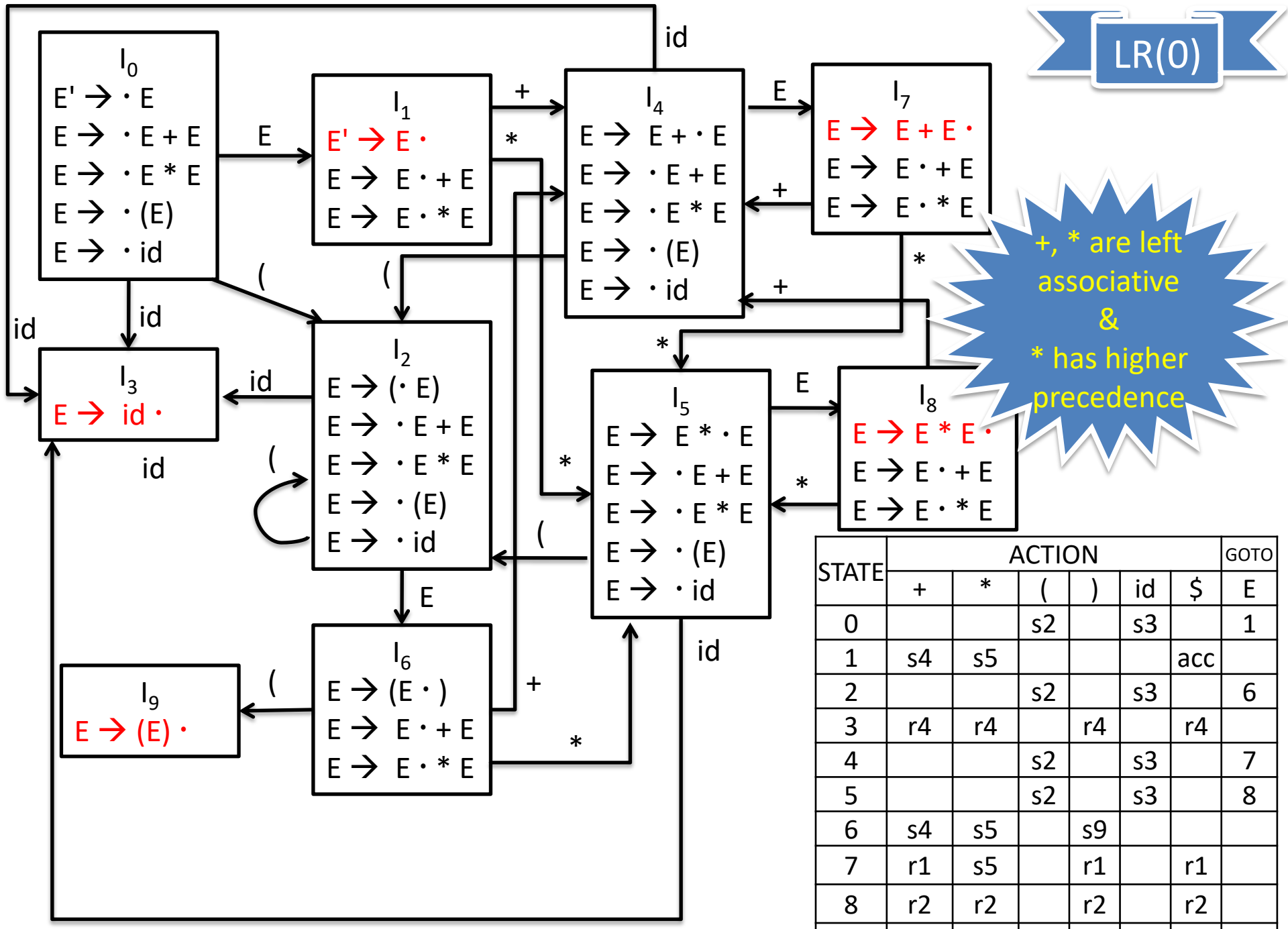


STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0			s2		s3		1
1	s4	s5				acc	
2			s2		s3		6
3	r4	r4		r4		r4	
4			s2		s3		7
5			s2		s3		8
6	s4	s5		s9			
7	r1	s5/r1		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	

- [7, +]: + operation is performed and again the + operation to be performed. [reduce]
- [7, *]: + operation is performed and then the * operation to be performed.
- [8, +]: * operation is performed and then the + operation is performed. [reduce]
- [8, *]: * operation is performed and then the * operation is performed. [reduce]



STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0			s2		s3		1
1	s4	s5				acc	
2			s2		s3		6
3	r4	r4		r4		r4	
4			s2		s3		7
5			s2		s3		8
6	s4	s5		s9			
7	r1	s5		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	



STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0			s2		s3		1
1	s4	s5				acc	
2			s2		s3		6
3	r4	r4		r4		r4	
4			s2		s3		7
5			s2		s3		8
6	s4	s5		s9			
7	r1	s5		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	

The “Dangling-Else” Ambiguity

- Consider the grammar for conditional statements:

  stmt → if expr then stmt else stmt

  | if expr then stmt

  | other

- Augmented Grammar

  S' → S

  S → iSeS

  S → i S

  S → a

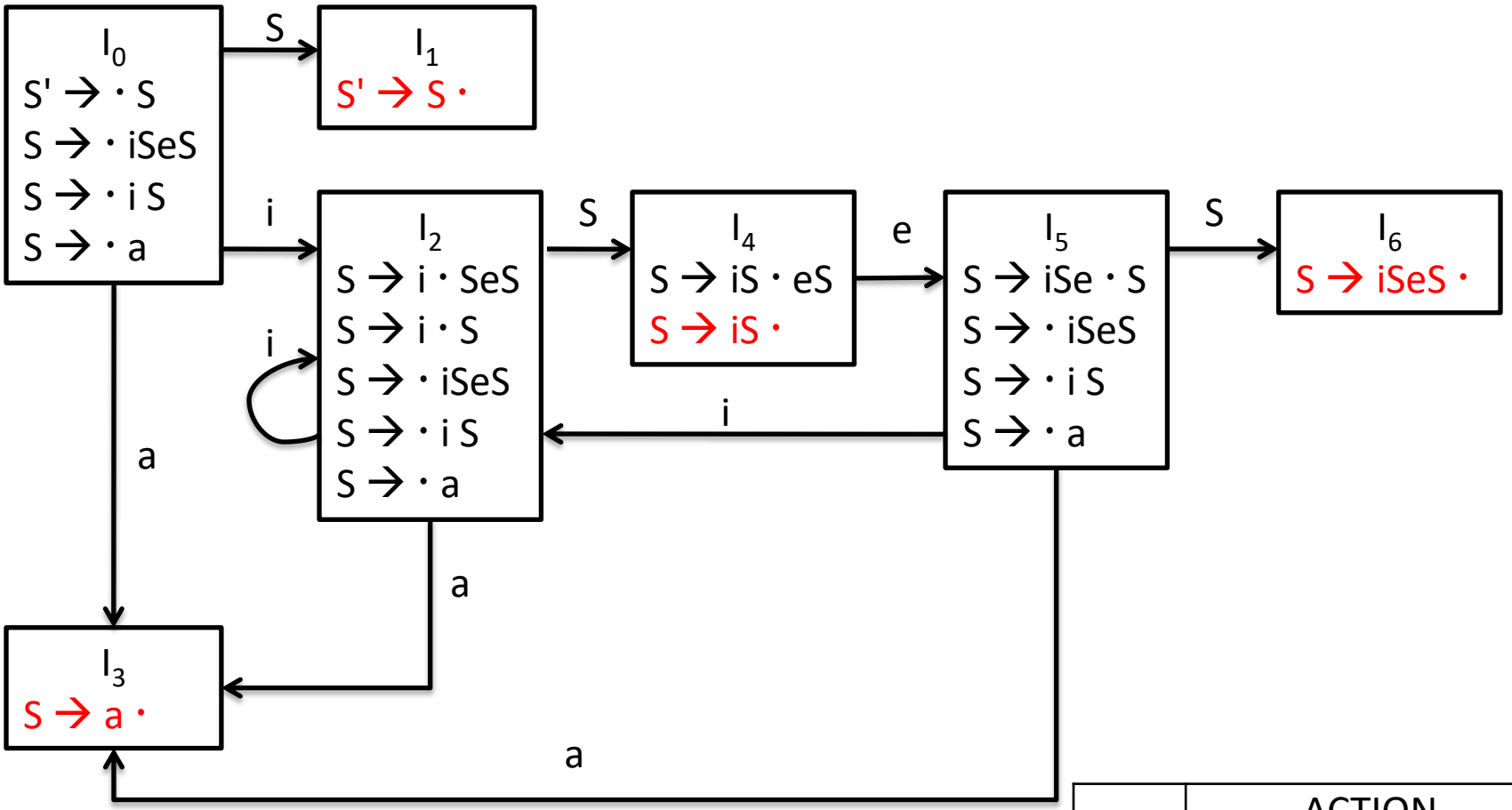
I_0

S' → · S

S → · iSeS

S → · i S

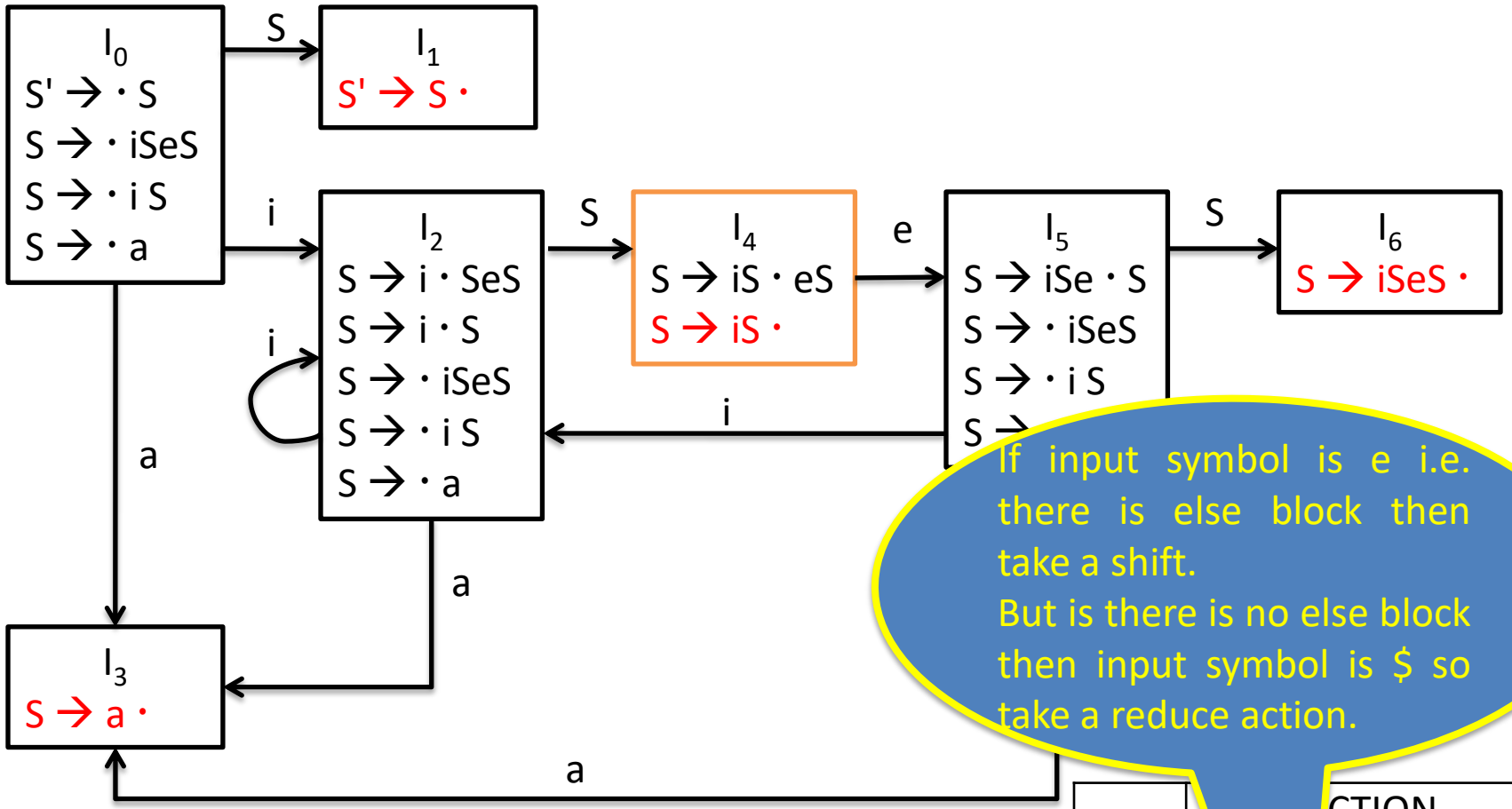
S → · a



1. S → iSeS
2. S → i S
3. S → a

FOLLOW(S) = {e, $}

STATE	ACTION				GOTO
	i	e	a	$	S
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		s5/r2		r2	
5	s2		s3		6
6		r1		r1	



1. S → iSeS
2. S → i S
3. S → a

FOLLOW(S) = {e, $}

STATE		CTION			GOTO
		e	a	$	S
0	s.		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		s5		r2	
5	s2		s3		6
6		r1		r1	

Error Recovery

- An LR parser will detect an error when it consults the parsing action table and finds an error entry.
- Errors are never detected by consulting the goto table.
- An LR parser will announce an error as soon as there is no valid continuation for the portion of the input thus far scanned.
- A **canonical LR parser** will not make even a single reduction before announcing an error.
- **SLR and LALR parsers** may make several reductions before announcing an error, but they will never shift an erroneous input symbol onto the stack.

Error Routines

STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0	e1	e1	s2		s3	e1	1
1	s4	s5				acc	
2	e1	e1	s2		s3	e1	6
3	r4	r4		r4		r4	
4	e1	e1	s2		s3	e1	7
5	e1	e1	s2		s3	e1	8
6	s4	s5		s9			
7	r1	s5		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	

e1:

- This routine is called from states 0, 2, 4 and 5, all of which expect the beginning of an operand, either an id or a left parenthesis.
- Instead, +, * , or the end of the input was found.
- push state 3 (the goto of states 0, 2, 4 and 5 on id);
- Issue diagnostic “missing operand”

Error Routines

STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0	e1	e1	s2	e2	s3	e1	1
1	s4	s5		e2		acc	
2	e1	e1	s2	e2	s3	e1	6
3	r4	r4		r4		r4	
4	e1	e1	s2	e2	s3	e1	7
5	e1	e1	s2	e2	s3	e1	8
6	s4	s5		s9			
7	r1	s5		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	

e2:

- Called from states 0, 1, 2, 4 and 5 on finding a right parenthesis
- remove the right parenthesis from the input;
- issue diagnostic “unbalanced right parenthesis”.

Error Routines

STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0	e1	e1	s2	e2	s3	e1	1
1	s4	s5	e3	e2	e3	acc	
2	e1	e1	s2	e2	s3	e1	6
3	r4	r4		r4		r4	
4	e1	e1	s2	e2	s3	e1	7
5	e1	e1	s2	e2	s3	e1	8
6	s4	s5	e3	s9	e3		
7	r1	s5		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	

e3:

- Called from states 1 or 6 when expecting an operator, and an id or left parenthesis is found.
- push state 4 (corresponding to symbol +) onto the stack;
- push state 5 (corresponding to symbol *) onto the stack;
- issue diagnostic “missing operator”.

Error Routines

STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0	e1	e1	s2	e2	s3	e1	1
1	s4	s5	e3	e2	e3	acc	
2	e1	e1	s2	e2	s3	e1	6
3	r4	r4		r4		r4	
4	e1	e1	s2	e2	s3	e1	7
5	e1	e1	s2	e2	s3	e1	8
6	s4	s5	e3	s9	e3	e4	
7	r1	s5		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	

e4:

- Called from state 6 when the end of the input is found.
- push state 9 (for a right parenthesis) onto the stack;
- issue diagnostic “missing right parenthesis”.

Error Routines

STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0	e1	e1	s2	e2	s3	e1	1
1	s4	s5	e3	e2	e3	acc	
2	e1	e1	s2	e2	s3	e1	6
3	r4	r4		r4		r4	
4	e1	e1	s2	e2	s3	e1	7
5	e1	e1	s2	e2	s3	e1	8
6	s4	s5	e3	s9	e3	e4	
7	r1	s5		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	

All other cells are filled with **respective reduce entries**.

Error Routines

STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0	e1	e1	s2	e2	s3	e1	1
1	s4	s5	e3	e2	e3	acc	
2	e1	e1	s2	e2	s3	e1	6
3	r4	r4	r4	r4	r4	r4	
4	e1	e1	s2	e2	s3	e1	7
5	e1	e1	s2	e2	s3	e1	8
6	s4	s5	e3	s9	e3	e4	
7	r1	s5	r1	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

All other cells are filled with **respective reduce entries**.

Error Routines

STATE	ACTION						GOTO
	+	*	(	)	id	$	E
0	e1	e1	s2	e2	s3	e1	1
1	s4	s5	e3	e2	e3	acc	
2	e1	e1	s2	e2	s3	e1	6
3	r4	r4	r4	r4	r4	r4	
4	e1	e1	s2	e2	s3	e1	7
5	e1	e1	s2	e2	s3	e1	8
6	s4	s5	e3	s9	e3	e4	
7	r1	s5	r1	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	