CLR(1) and LALR(1) Parsers

C.Naga Raju
B.Tech(CSE),M.Tech(CSE),PhD(CSE),MIEEE,MCSI,MISTE
Professor
Department of CSE
YSR Engineering College of YVU
Proddatur

Contents

- Limitations of SLR(1) Parser
- Introduction to CLR(1) Parser
- Animated example
- Limitation of CLR(1)
- LALR(1) Parser Animated example
- GATE Problems and Solutions

Drawbacks of SLR(1)

- The SLR Parser discussed in the earlier class has certain flaws.
- 1.On single input, State may be included a Final Item and a Non- Final Item. This may result in a Shift-Reduce Conflict .
- 2.A State may be included Two Different Final Items. This might result in a Reduce-Reduce Conflict

- 3.SLR(1) Parser reduces only when the next token is in Follow of the left-hand side of the production.
- 4.SLR(1) can reduce shift-reduce conflicts but not reduce-reduce conflicts
- These two conflicts are reduced by CLR(1) Parser by keeping track of lookahead information in the states of the parser.
- This is also called as **LR(1) grammar**

CLR(1) Parser

- LR(1) Parser greatly increases the strength of the parser, but also the size of its parse tables.

- The LR(1) techniques does not rely on FOLLOW sets, but it keeps the Specific Look-ahead with each item.

CLR(1) Parser

❖ LR(1) Parsing configurations have the general form:

A –> X1…Xi • Xi+1…Xj , a

❖ The Look Ahead Component **'a'** represents a **possible look-ahead** after the entire right-hand side has been matched

❖ The **є** appears as look-ahead **only for the augmenting production** because there is **no look-ahead** after the end-marker

Prof.C.NagaRaju YSREC of YVU
9949218570

LR(1) Item

- The LR(k) Items are used to represent the set of possible states in a parser
- An LR(k) item is a **pair [α, β],**

$$\boxed{\mathbf{A \rightarrow \alpha.\beta,a}}$$ **where**

  - ✓ **a** is the look-head of the LR(1) item (**a** is a terminal or end-marker.)
  - ✓ α is a production from G with ● at some position in the RHS
  - ✓ β is a Lookahead String containing k symbols (Terminals or EOF)

LR(1) Item

- If β is not empty string and contains terminal symbols , then the look-head does not have any affect. If it is Non terminal symbol it should have effects.

$$[A \rightarrow \alpha \bullet X\beta,\ a] \xrightarrow{X} [A \rightarrow \alpha X \bullet \beta,\ a]$$

- if β is empty string then reduce with $A \rightarrow \alpha$,only if the next input symbol is **a**

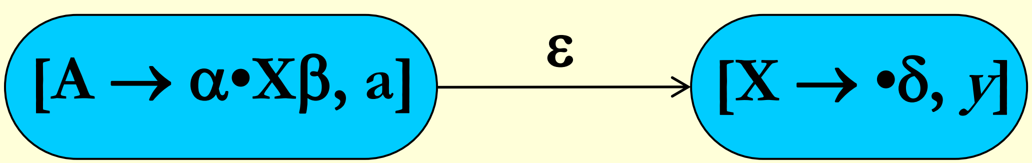
$$q_0 \xrightarrow{\varepsilon} [S \rightarrow \bullet\alpha,\ \varepsilon]$$

- A State will contain $\mathbf{A \rightarrow \alpha .,a_1}$ where $\{\mathbf{a_1,\ldots,a_n}\} \subseteq$ **FOLLOW(A)**

$$\mathbf{A \rightarrow \alpha .,a_1 \;\ldots\ldots\ldots\; A \rightarrow \alpha .,a_n}$$

LR(1) Closure Items

- Every LR(1) item in I is in closure(I)
- If **$A\rightarrow\alpha.B\beta,a$** in closure(I) and **$B\rightarrow\gamma$** is a production rule of G; then $B\rightarrow.\gamma,b$ will be in the **closure(I)** for each terminal b in FIRST(βa)
- For every LR(1) item [A → a•Xb, a] and production X → δ and every y in FIRST(βx)



Example: CLR(1) Parser

- Construct the **CLR(1) Parser** for the Following **Grammar**

Context Free Grammar:

S → CC

C → cC

C → d

Step 1: Define a Augmented Grammar

Context Free Grammar:

$S \rightarrow CC$

$C \rightarrow cC$

$C \rightarrow d$

.

Augmented Grammar:

$S' \rightarrow \bullet\ S\$$

$S \rightarrow \bullet CC$

$C \rightarrow \bullet cC$

$C \rightarrow \bullet d$

Step2 : Constructing the LR(1) Items

Constructing the LR(1) Closure Items

Closure1(S'→ • S$, {Є})

A → α.β,a

S' → ·S,{$}

S → • CC,{ $}

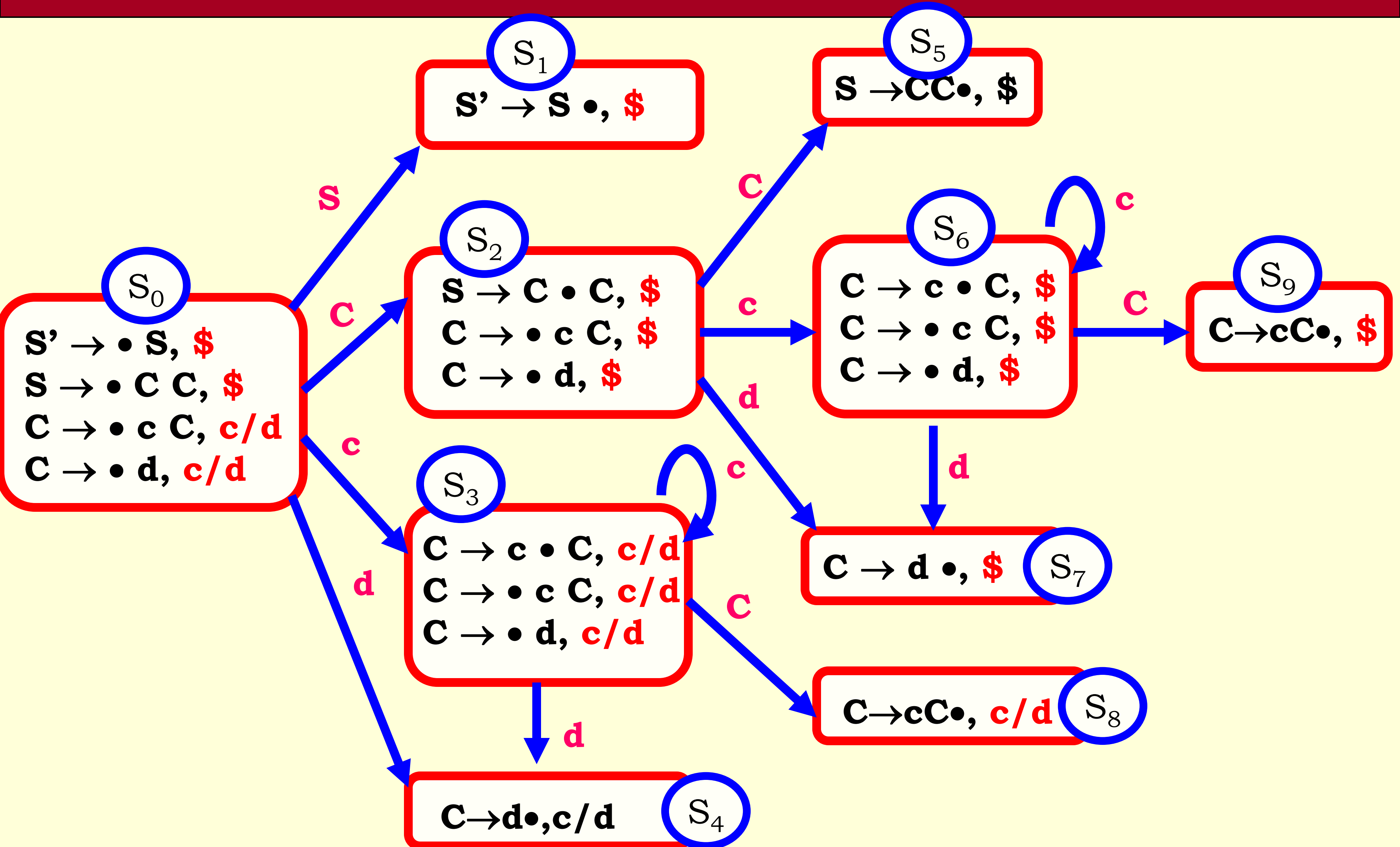
C → • c C, {c/d}
C → • d, {c/d}

S' → • S$
S → •CC
C → •cC
C → •d

Closure1(S→ • E$, **{Є})=**
{
S' →•S, {$}
S →•CC, {$}
C →•cC, {c/d}
C →•d, {c/d}
}

Prof.C.NagaRaju YSREC of YVU
9949218570

Construction of DFA for CLR(1) Items



Constructing the LR(1) Closure Items

S_0

S' → • S, $
S → • C C, $
C → • c C, c/d
C → • d, c/d

S_1: goto(S_0, S)

S_1

S' → S •, $

S_0

S' → • S, $

S → • C C, $

C → • c C, c/d

C → • d, c/d

S_2: goto(S_0, C)

S_1

S' → S •, $

S_2

S → C • C, $
C → • c C, $
C → • d, $

S_0

S' → • S, $
S → • C C, $
C → • c C, c/d
C → • d, c/d

S_3: goto(S_0, c)

S_1

S' → S •, $

S_2

S → C • C, $
C → • c C, $
C → • d, $

S_0

S' → • S, $
S → • C C, $
C → • c C, c/d
C → • d, c/d

S_3

C → c • C, c/d
C → • c C, c/d
C → • d, c/d

S_4: goto(S_0, d)

S_0
- $S' \rightarrow \bullet\, S,\ \$$
- $S \rightarrow \bullet\, C\, C,\ \$$
- $C \rightarrow \bullet\, c\, C,\ c/d$
- $C \rightarrow \bullet\, d,\ c/d$

S_1
- $S' \rightarrow S\, \bullet,\ \$$

S_2
- $S \rightarrow C \bullet C,\ \$$
- $C \rightarrow \bullet\, c\, C,\ \$$
- $C \rightarrow \bullet\, d,\ \$$

S_3
- $C \rightarrow c \bullet C,\ c/d$
- $C \rightarrow \bullet\, c\, C,\ c/d$
- $C \rightarrow \bullet\, d,\ c/d$

S_4
- $C \rightarrow d\, \bullet,\ c/d$

Prof.C.NagaRaju YSREC of YVU
9949218570

S_5: goto(S_2, C)

S_1

S' → S •, $

S_5

(S → C C •, $)

S_2

S → C • C, $
C → • c C, $
C → • d, $

S_0

S' → • S, $
S → • C C, $
C → • c C, c/d
C → • d, c/d

S_3

C → c • C, c/d
C → • c C, c/d
C → • d, c/d

S_4

C → d •, c/d

S_6: goto(S_2, c)

S_0

S' → • S, $
S → • C C, $
C → • c C, c/d
C → • d, c/d

S_1

S' → S •, $

S_2

S → C • C, $
C → • c C, $
C → • d, $

S_3

C → c • C, c/d
C → • c C, c/d
C → • d, c/d

S_4

C → d •, c/d

S_5

(S → C C •, $)

S_6

C → c • C, $
C → • c C, $
C → • d, $

S_7: goto(S_2, c)

S_1: S' → S •, $

S_5: (S → C C •, $)

S_2:
S → C • C, $
C → • c C, $
C → • d, $

S_6:
C → c • C, $
C → • c C, $
C → • d, $

S_0:
S' → • S, $
S → • C C, $
C → • c C, c/d
C → • d, c/d

S_3:
C → c • C, c/d
C → • c C, c/d
C → • d, c/d

S_7: C → d •, $

S_4: C → d •, c/d

S_8: goto(S_3, C)

S_1
S' → S •, $

S_5
(S → C C •, $)

S_2
S → C • C, $
C → • c C, $
C → • d, $

S_6
C → c • C, $
C → • c C, $
C → • d, $

S_0
S' → • S, $
S → • C C, $
C → • c C, c/d
C → • d, c/d

S_3
C → c • C, c/d
C → • c C, c/d
C → • d, c/d

S_7
C → d •, $

S_8
C → c C •, c/d

S_4
C → d •, c/d

S_9: goto(S_6, C)

S_0

$S' \rightarrow \bullet\ S,\ \$$
$S \rightarrow \bullet\ C\ C,\ \$$
$C \rightarrow \bullet\ c\ C,\ c/d$
$C \rightarrow \bullet\ d,\ c/d$

S_1

$S' \rightarrow S\ \bullet,\ \$$

S_2

$S \rightarrow C\ \bullet\ C,\ \$$
$C \rightarrow \bullet\ c\ C,\ \$$
$C \rightarrow \bullet\ d,\ \$$

S_3

$C \rightarrow c\ \bullet\ C,\ c/d$
$C \rightarrow \bullet\ c\ C,\ c/d$
$C \rightarrow \bullet\ d,\ c/d$

S_4

$C \rightarrow d\ \bullet,\ c/d$

S_5

$(S \rightarrow C\ C\ \bullet,\ \$)$

S_6

$C \rightarrow c\ \bullet\ C,\ \$$
$C \rightarrow \bullet\ c\ C,\ \$$
$C \rightarrow \bullet\ d,\ \$$

S_7

$C \rightarrow d\ \bullet,\ \$$

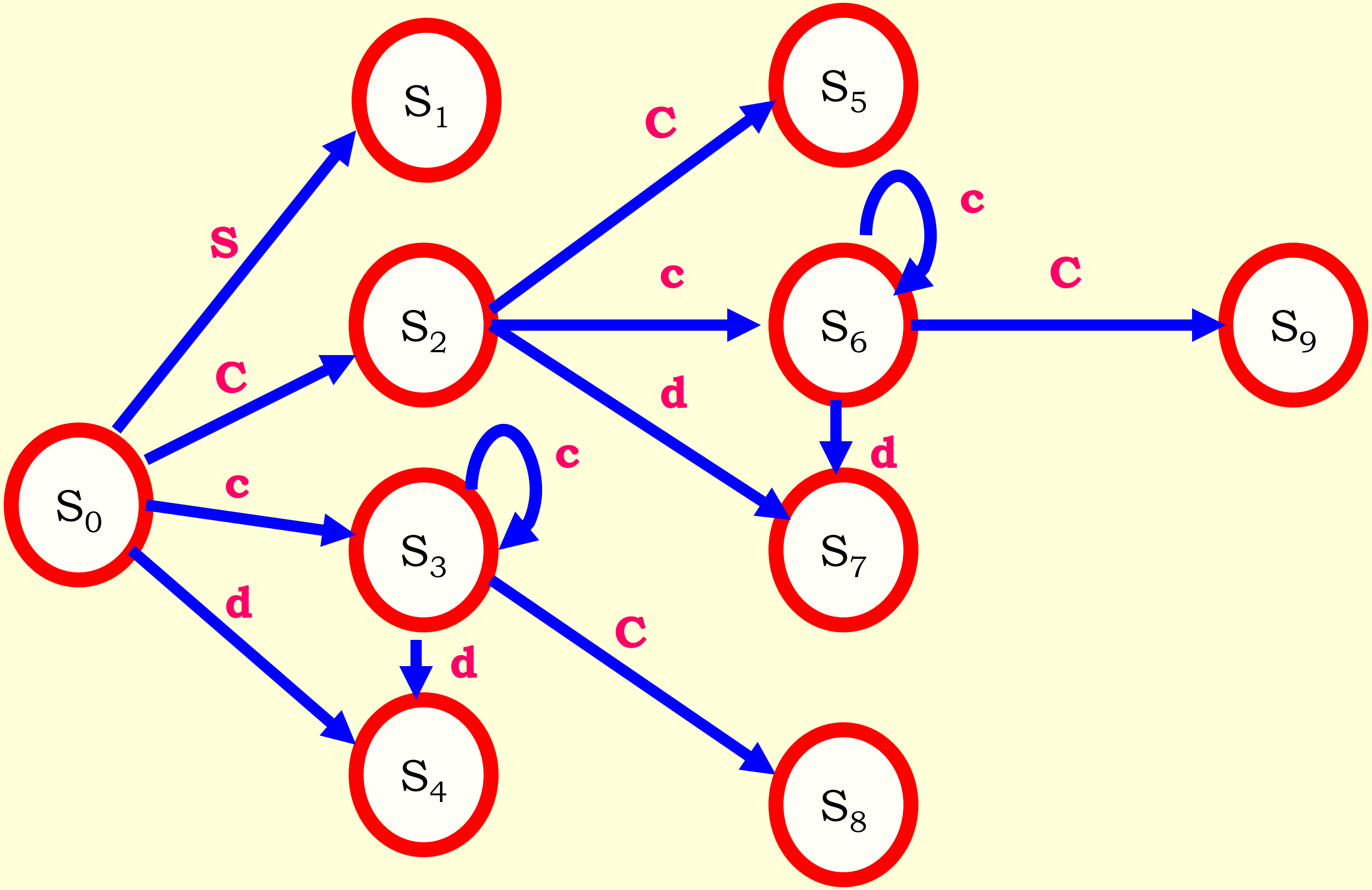
S_8

$C \rightarrow c\ C\ \bullet,\ c/d$

S_9

$C \rightarrow c\ C\bullet,\ \$$

Construction of DFA for CLR(1) Items



Construction of Follow Function

$S' \rightarrow S\$$

$S \rightarrow C\ C$ (r1)

$C \rightarrow c\ C$ (r2)

$C \rightarrow d$ (r3)

Follow (S) = { $ }

Follow (C) = { $,c, d }

Prof.C.NagaRaju YSREC of YVU
9949218570

Constructing the CLR(1) Parsing Table

States	Input				
	Action Part			Goto Part	
	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		

- In the LR parsing table of a grammar G has no Conflict, Therefore the Grammar is called LR(1) Grammar.

LALR(1) Parser

- An LALR(1) parser is an LR(1) parser in which all states that differ only in the lookahead components of the configurations are **merged and** union of lookaheads
- LALR (Look Ahead LR) Parser reduces the number of states to the same as SLR(1) Parser
- LALR (Look Ahead LR) Parser still retains some of the power of the LR(1) lookaheads.

Algorithm :Construction of LALR(1) Parser

1. Construct all canonical LR(1) states.
2. Merge those states that are identical if the lookaheads are ignored, i.e., two states being merged must have the same number of items and the items have the same core.
3. The Goto Function for the new LALR(1) state is the union of the merged states.
4. The action and goto entries are constructed from the LALR(1) states as for the canonical LR(1) parser.

Identifying the Common Core & Merging

S_6

$C \rightarrow c \bullet C,\ \$$

$C \rightarrow \bullet c\, C,\ \$$

$C \rightarrow \bullet d,\ \$$

S_3

$C \rightarrow c \bullet C,\ c/d$

$C \rightarrow \bullet c\, C,\ c/d$

$C \rightarrow \bullet d,\ c/d$

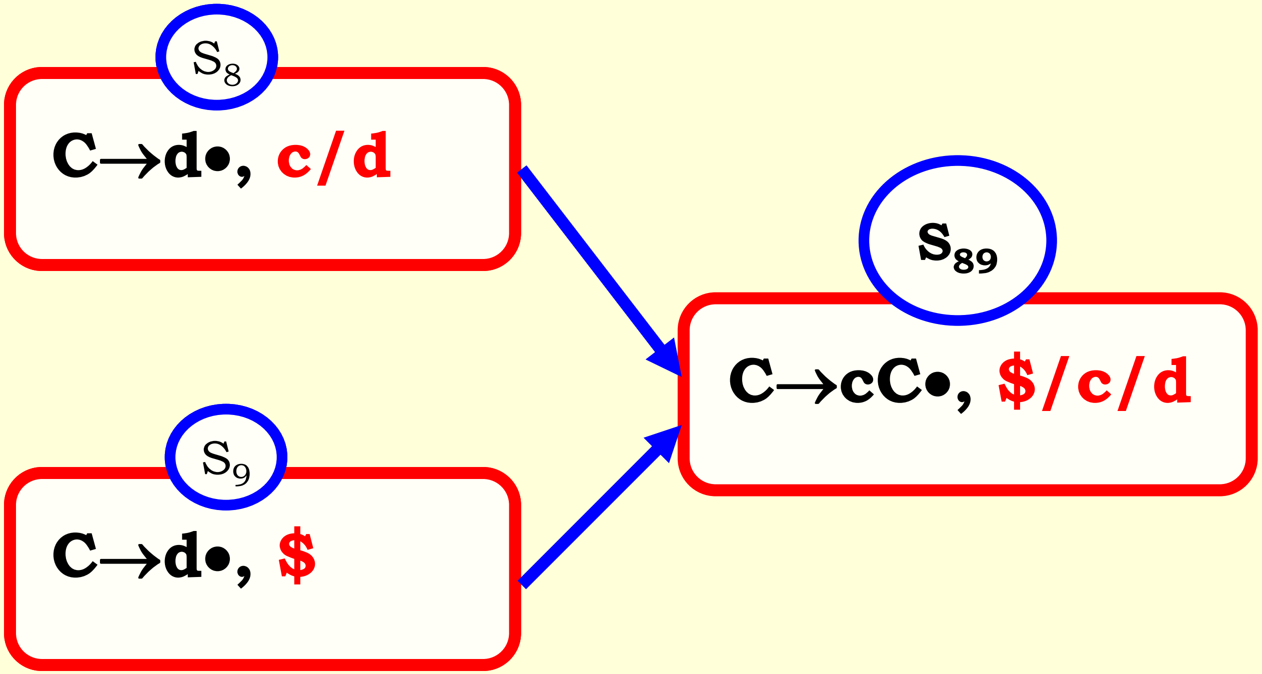
S_{36}

$C \rightarrow c \bullet C,\ c/d/\$$

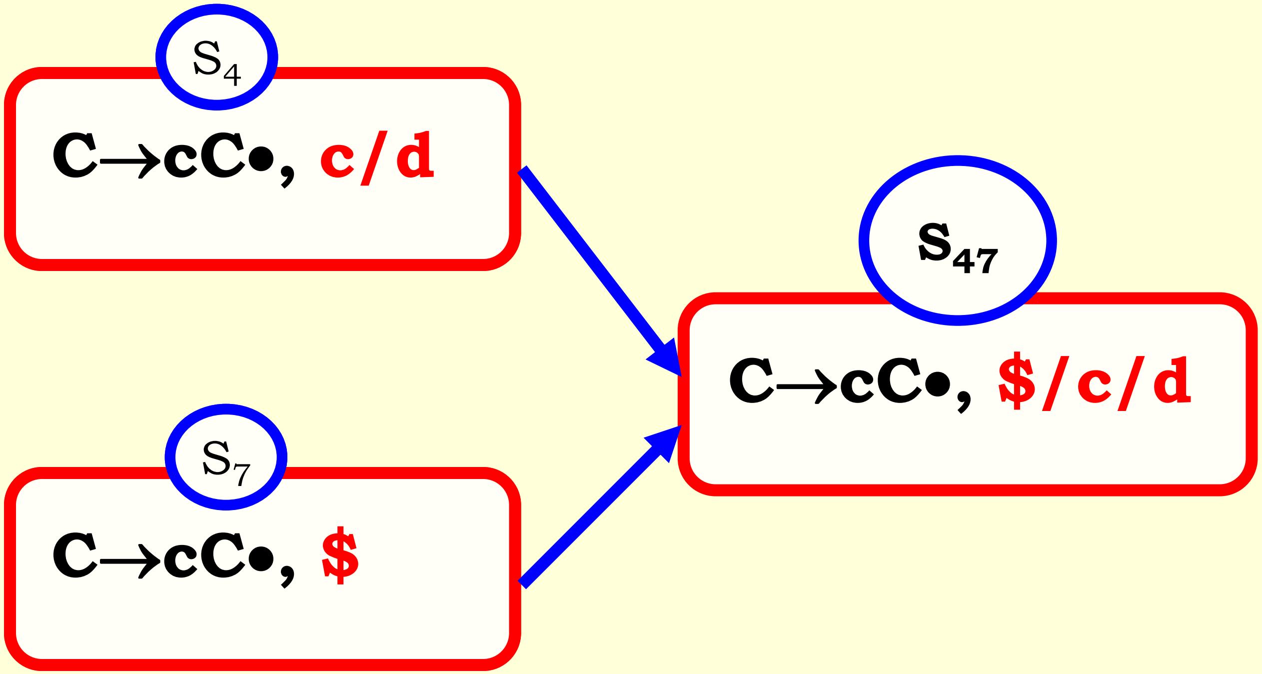
$C \rightarrow \bullet c\, C,\ c/d/\$$

$C \rightarrow \bullet d,\ c/d/\$$

Identifying the Common Core & Merging



Identifying the Common Core & Merging



Construction of DFA for LALR(1) Items

S_0

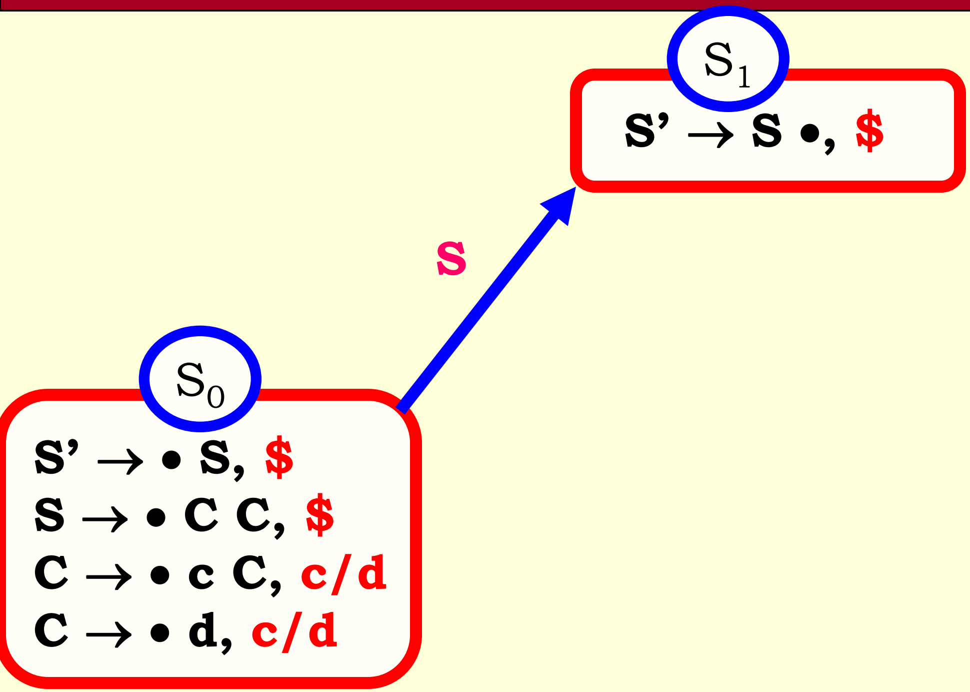
S' → • S, $

S → • C C, $

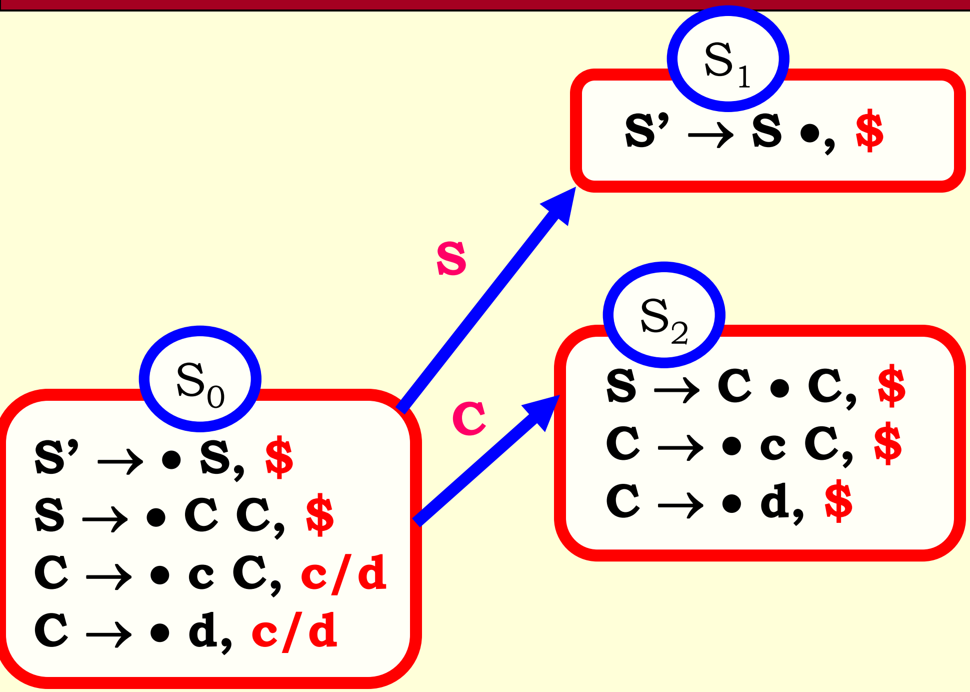
C → • c C, c/d

C → • d, c/d

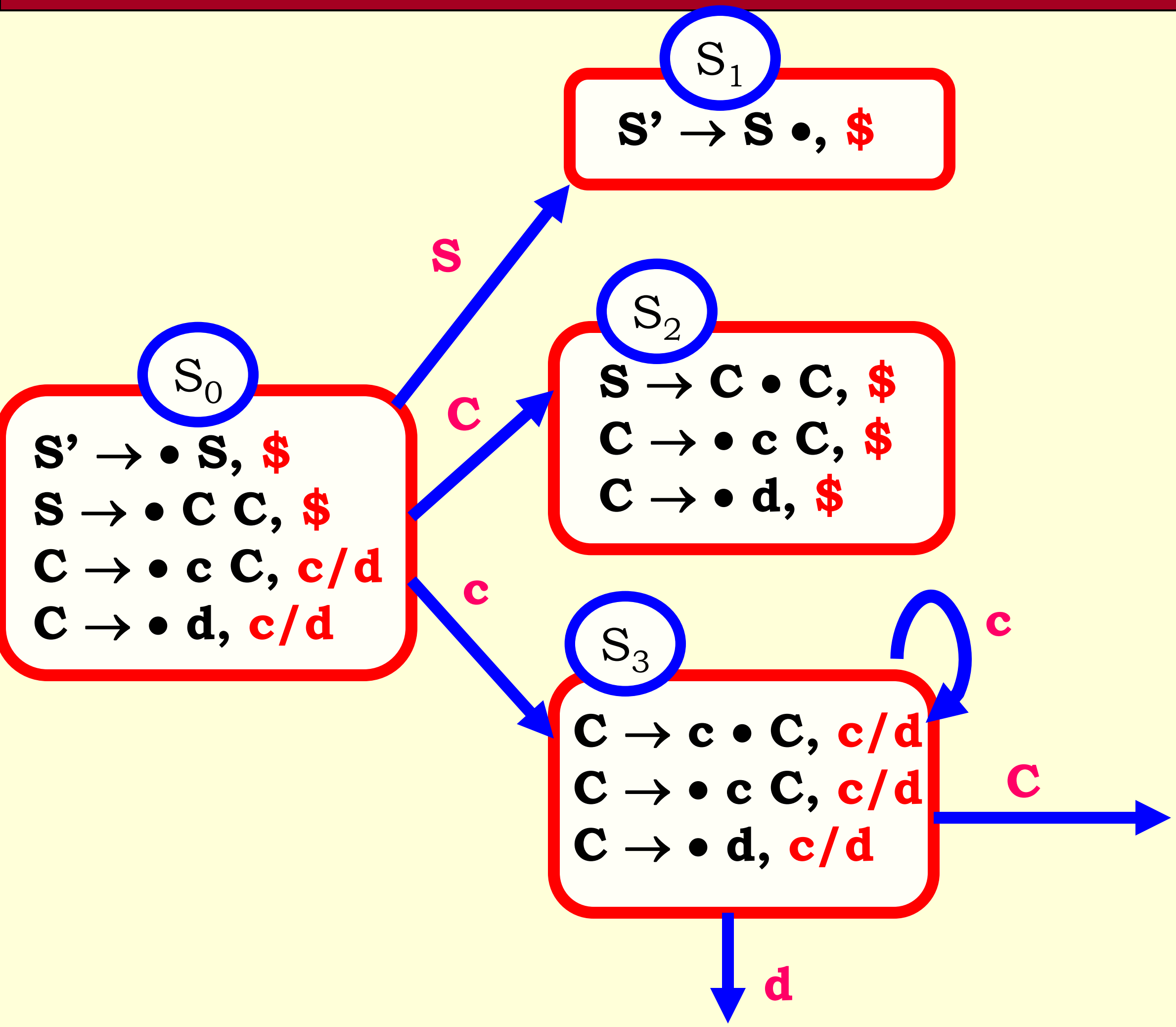
Construction of DFA for LALR(1) Items



Construction of DFA for LALR(1) Items

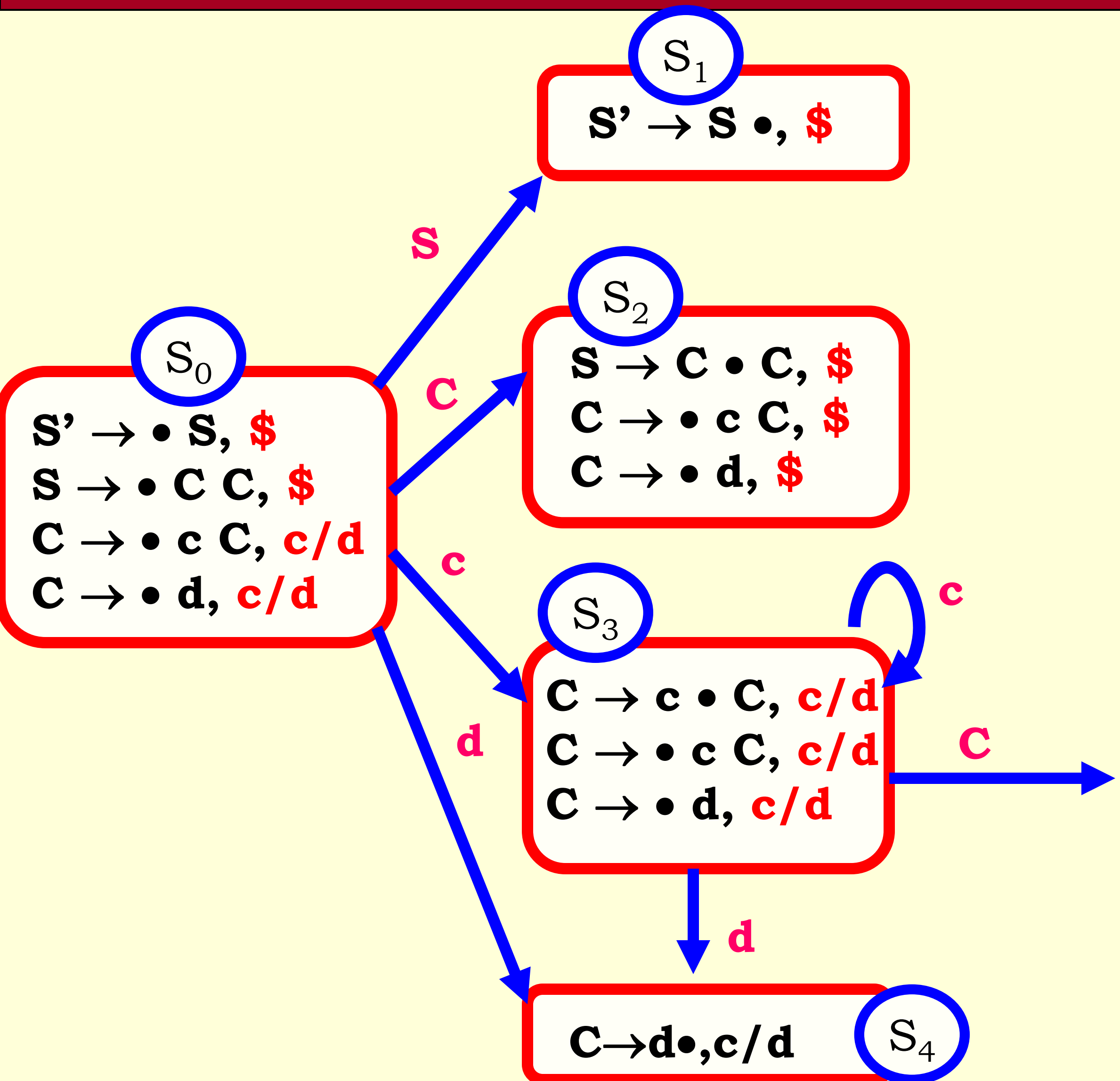


Construction of DFA for LALR(1) Items

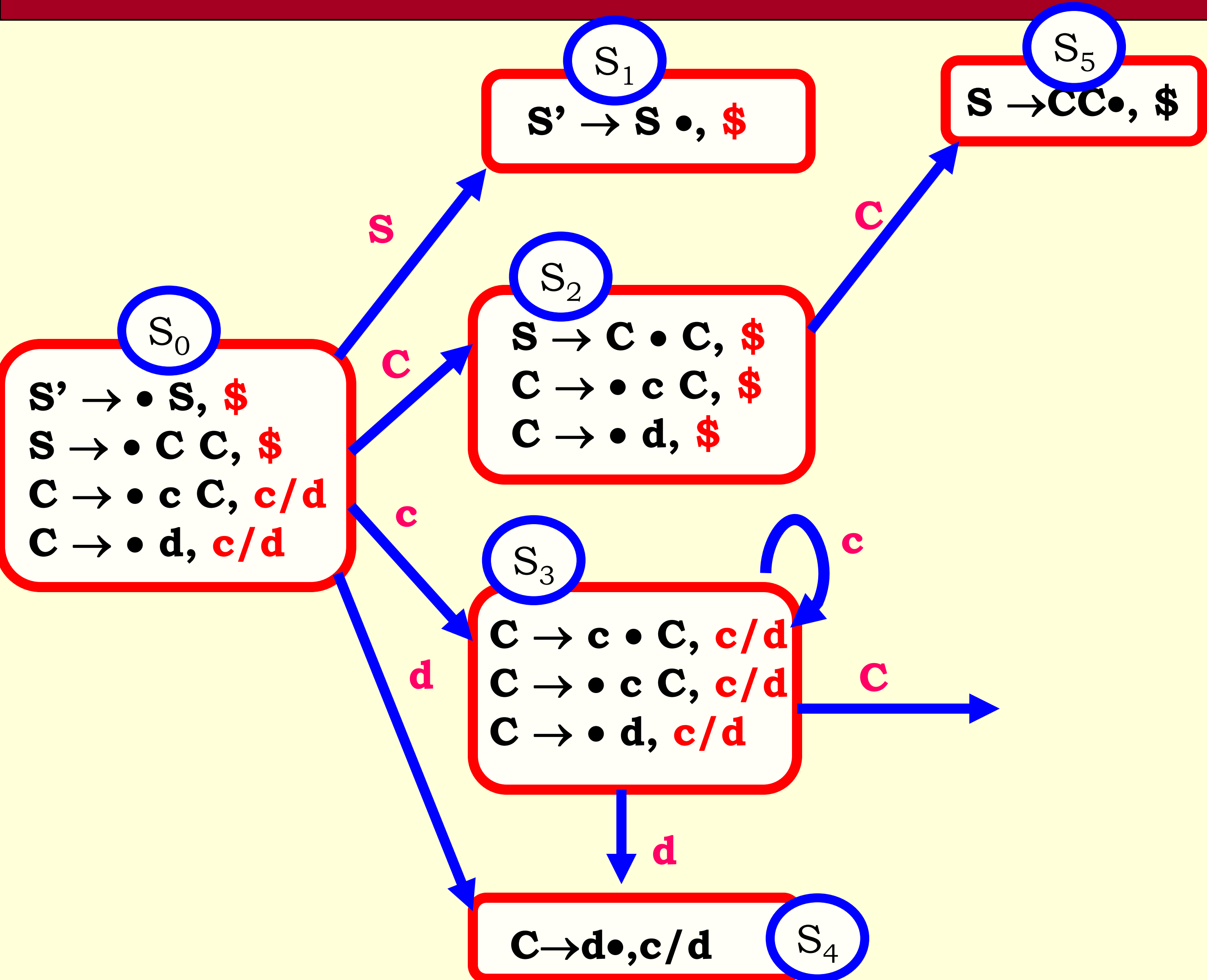


Prof.C.NagaRaju YSREC of YVU
9949218570

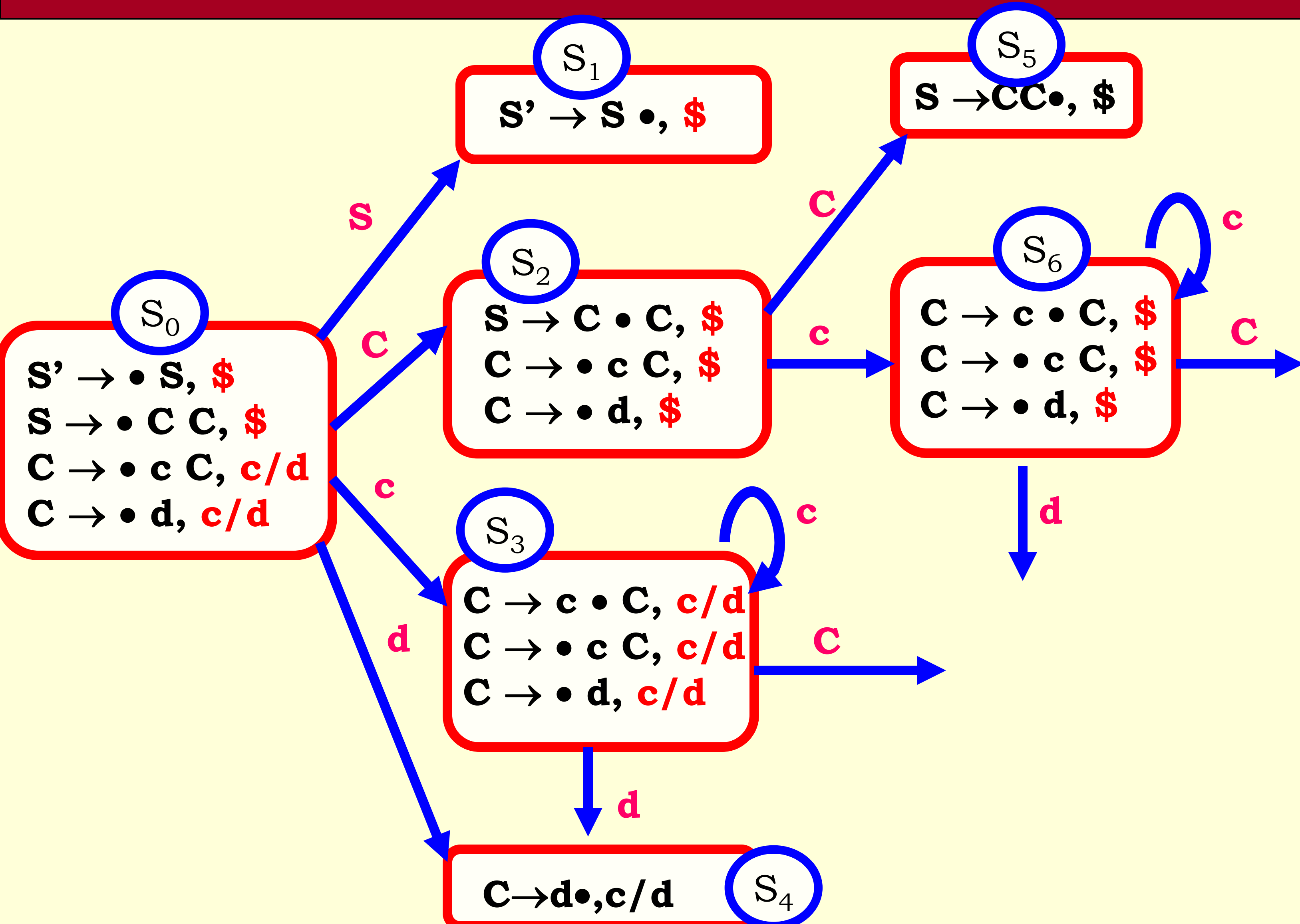
Construction of DFA for LALR(1) Items



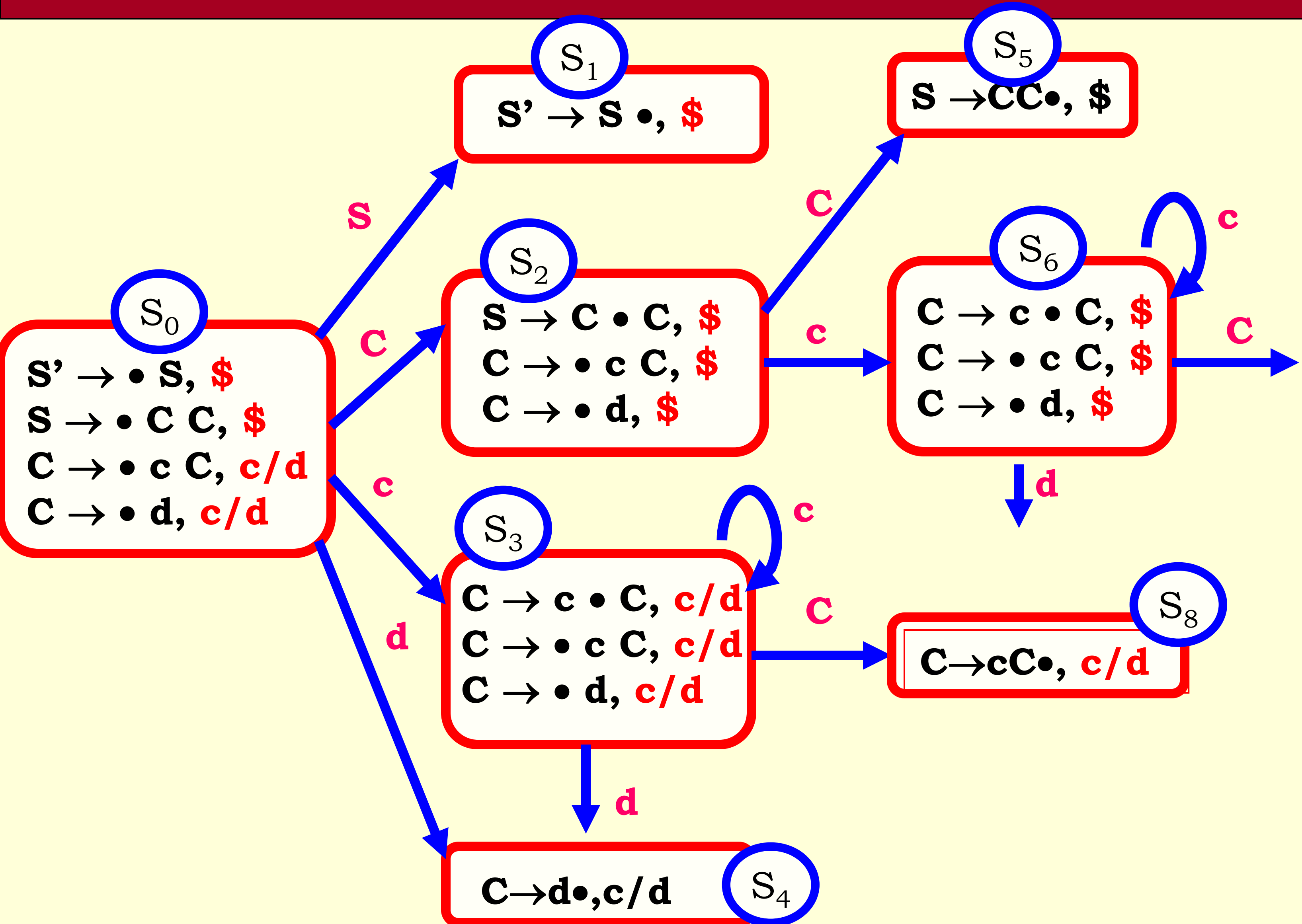
Construction of DFA for LALR(1) Items



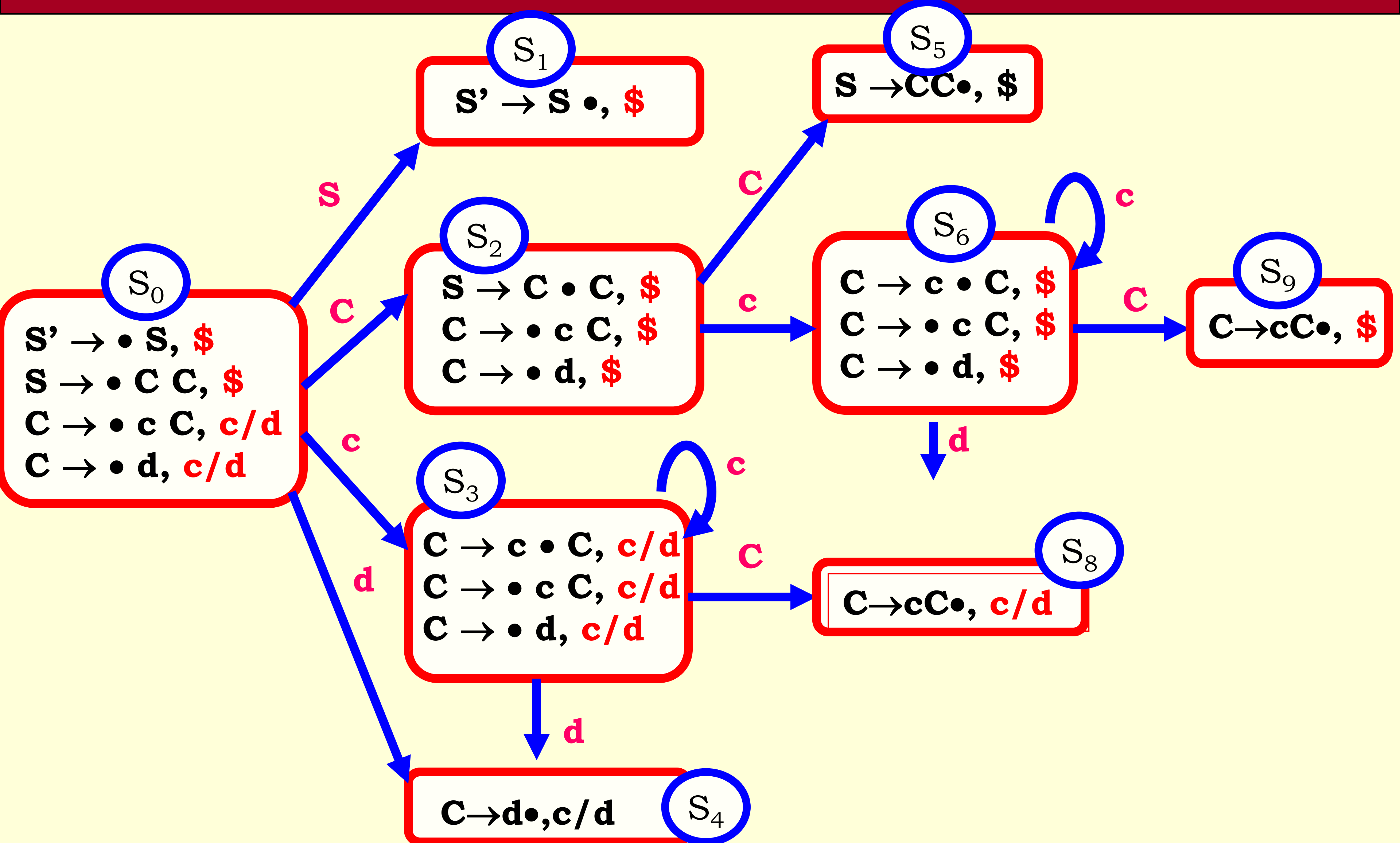
Construction of DFA for LALR(1) Items



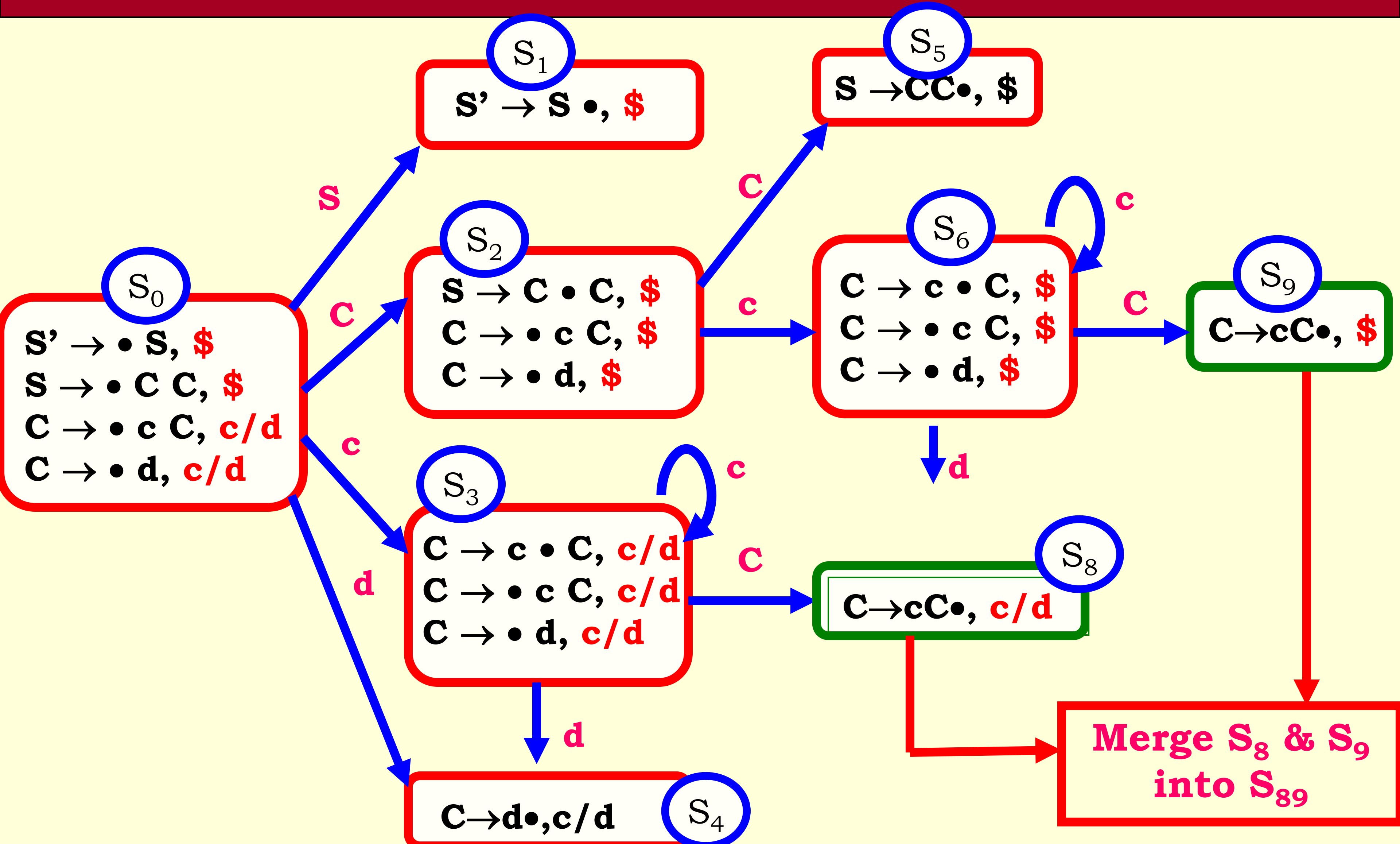
Construction of DFA for LALR(1) Items



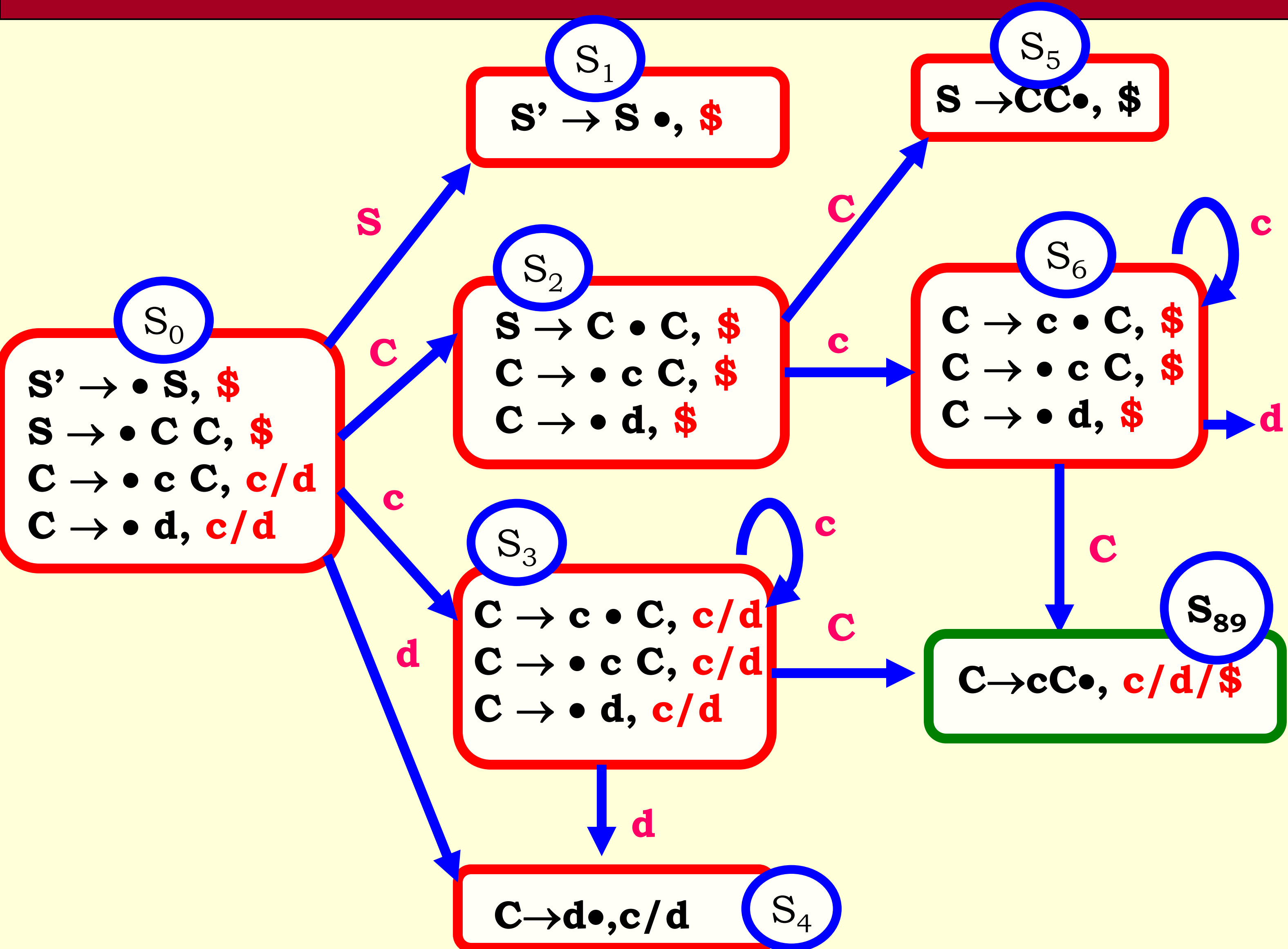
Construction of DFA for LALR(1) Items



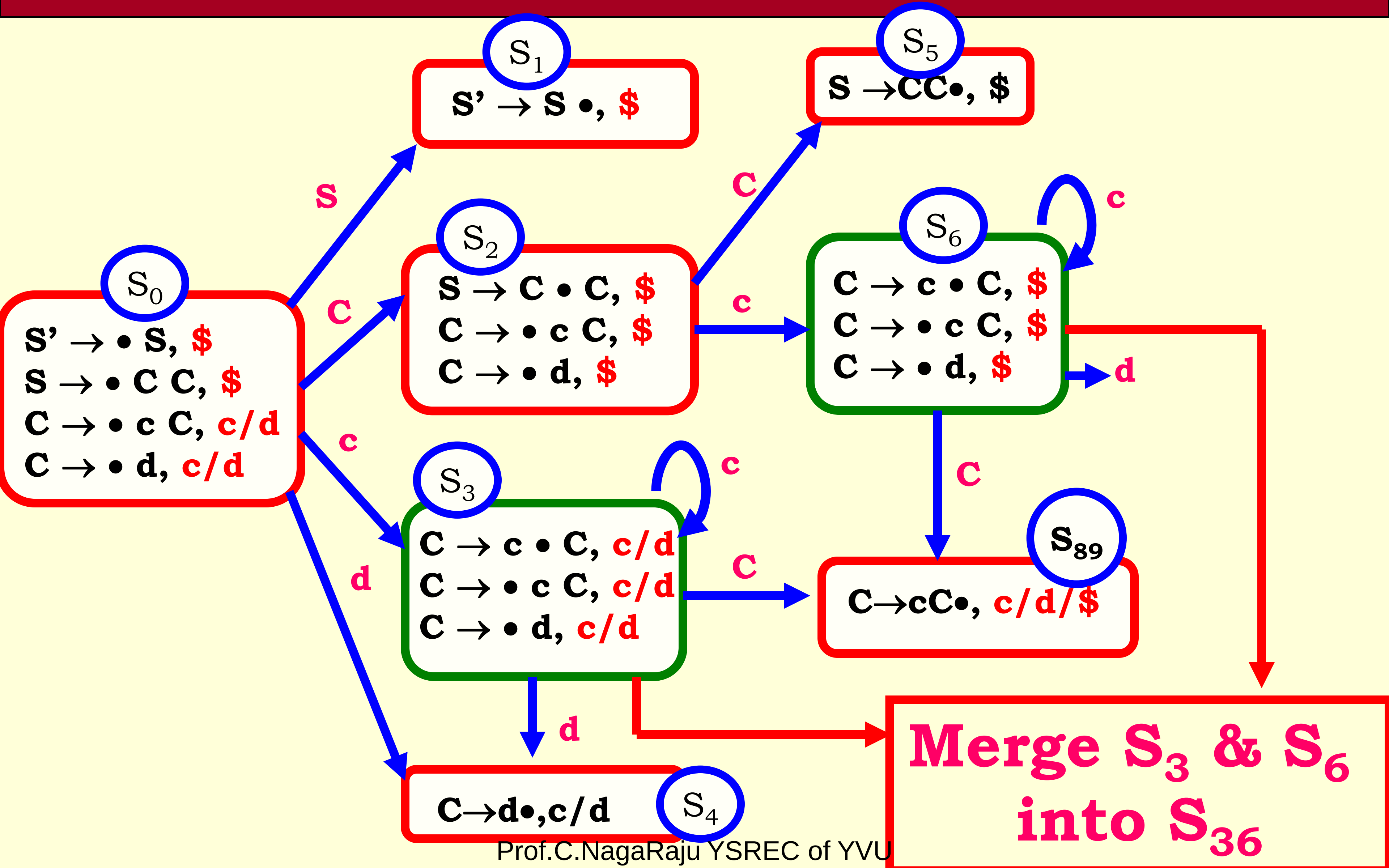
Construction of DFA for LALR(1) Items



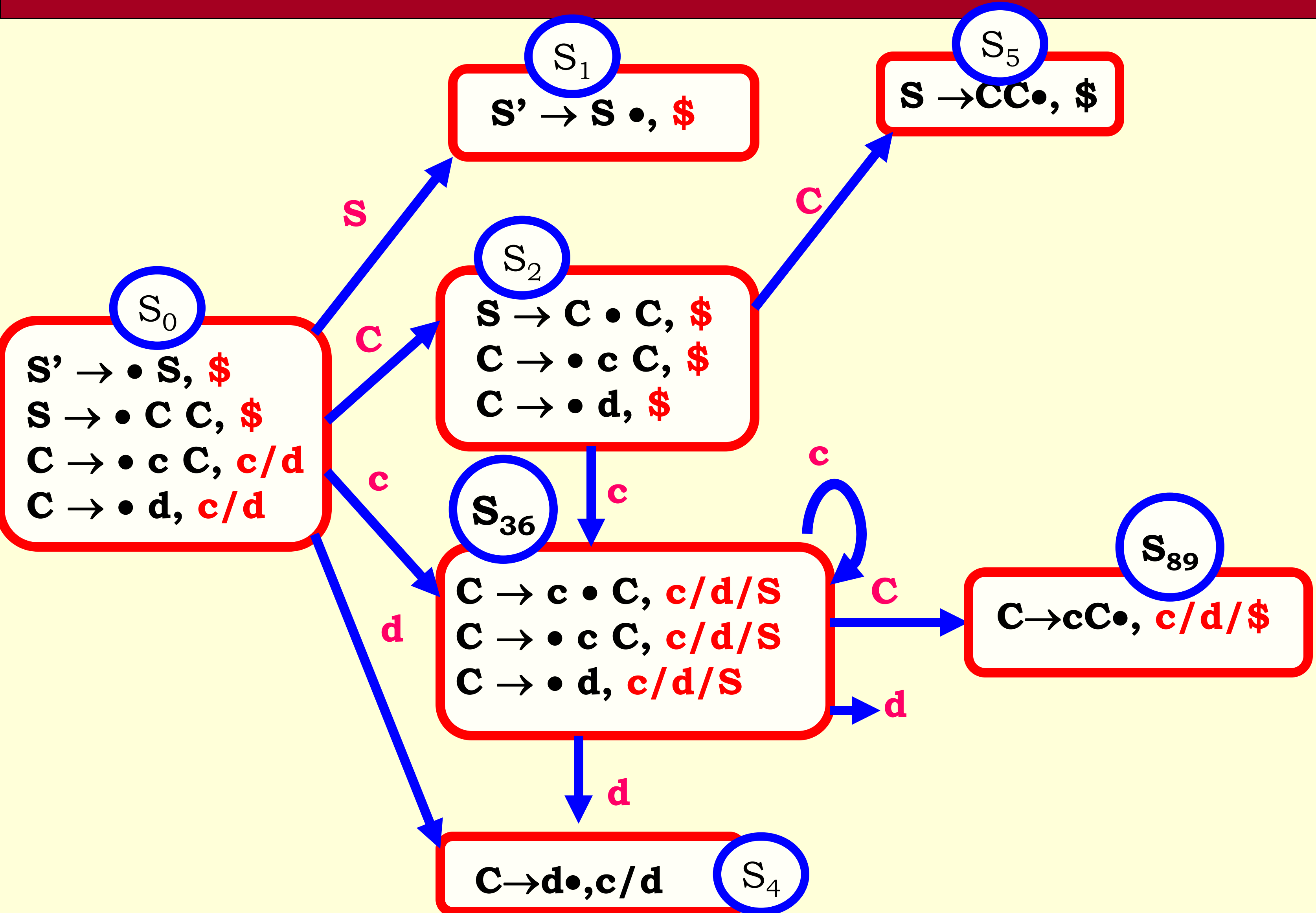
Construction of DFA for LALR(1) Items



Construction of DFA for LALR(1) Items

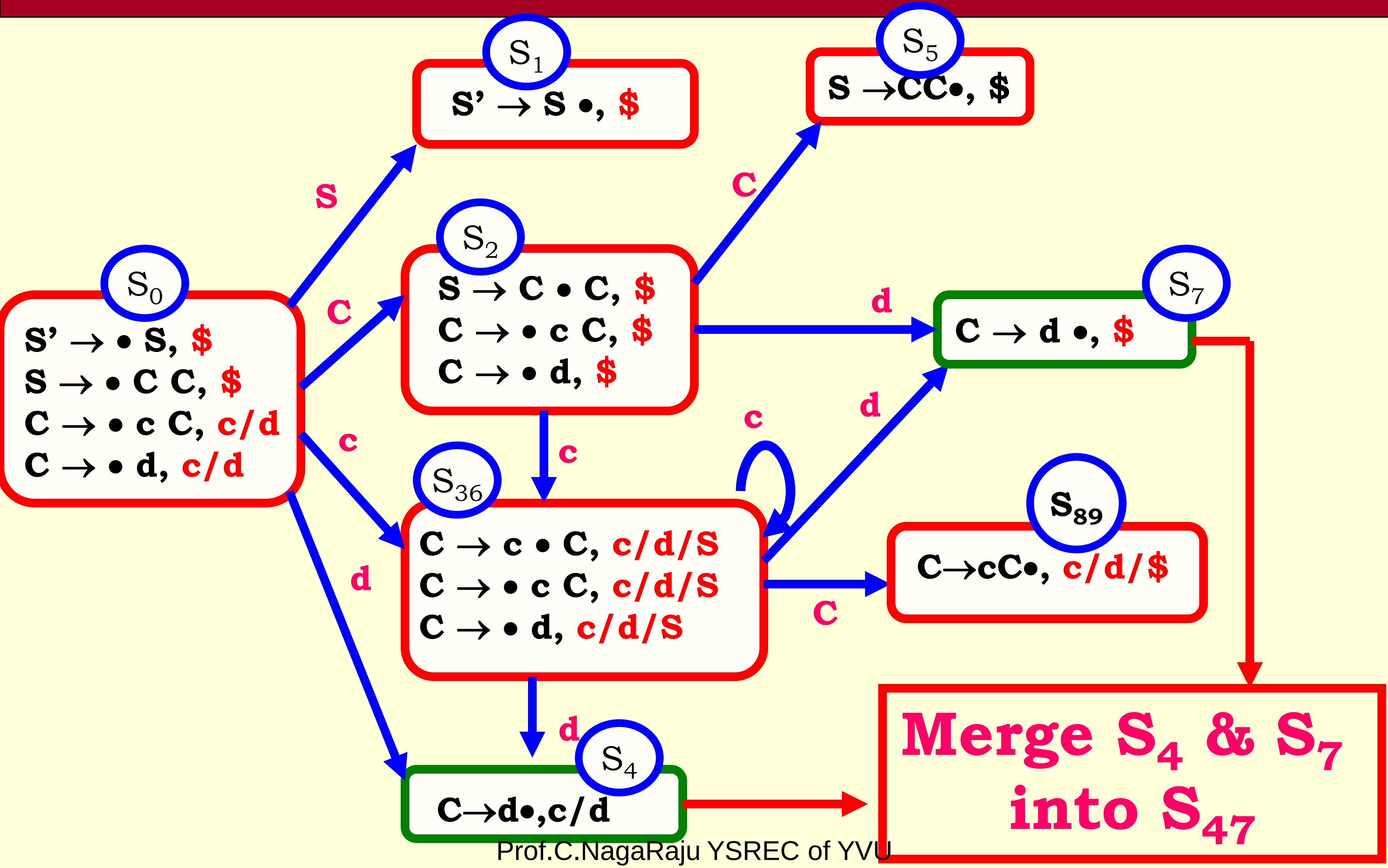


Construction of DFA for LALR(1) Items

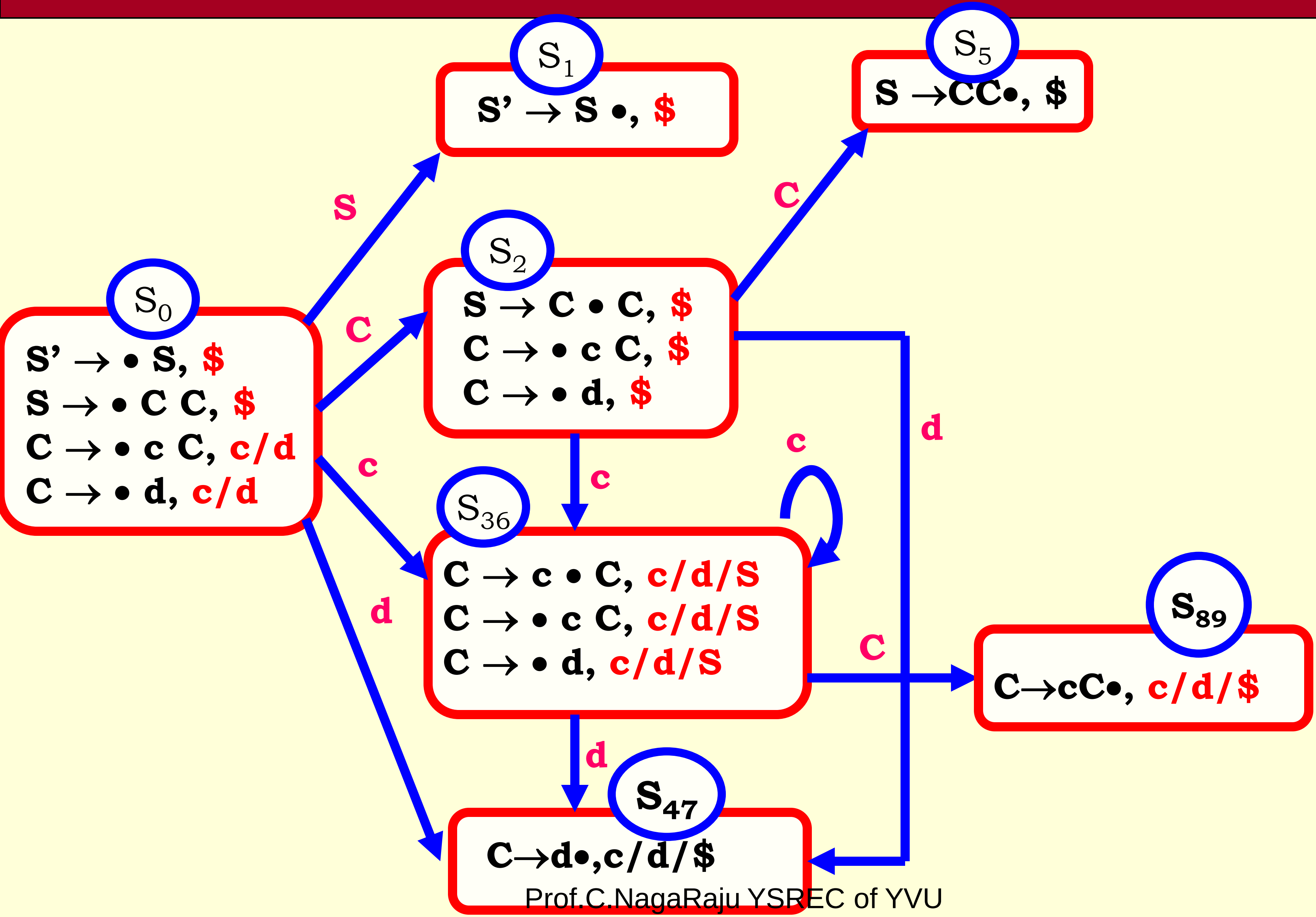


Prof.C.NagaRaju YSREC of YVU
9949218570

Construction of DFA for LALR(1) Items



Construction of DFA for LALR(1) Items



Constructing the CLR(1) Parsing Table

States	Input				
	Action Part			Goto Part	
	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		

Shift/Reduce Conflict

- We say that we cannot get a Shift/Reduce Conflict during the Merging process for the creation of the states of a LALR parser.
- Assume that we may get a Shift/Reduce Conflict. In this case, a state of LALR parser must have:

$$\mathbf{A \to \alpha.,a \quad and \quad B \to \beta.a\gamma,b}$$

- This means that a state of the canonical LR(1) parser must have:

$$\mathbf{A \to \alpha.,a \quad and \quad B \to \beta.a\gamma,c}$$

Reduce/Reduce Conflict

- we may get Reduce/Reduce Conflict during the Merging process for the creation of the states of a LALR parser.

I_1 : $A \rightarrow \alpha.,a$
$B \rightarrow \beta.,b$

I_2: $A \rightarrow \alpha.,b$
$B \rightarrow \beta.,c$

$$\Downarrow$$

I_{12}: $A \rightarrow \alpha.,a/b$
$B \rightarrow \beta.,b/c$

Reduce/Reduce Conflict

Prof.C.NagaRaju YSREC of YVU
9949218570

Conclusion

- A Grammar that is SLR(1) is definitely LALR(1).
- A Grammar that is not SLR(1) may or may not be LALR(1) depending on whether the more precise lookaheads resolve the SLR(1) conflicts.
- **LALR(1) Parser** has proven to be the most used variant of the LR family.
- LALR(1) parsers and most programming language constructs can be described with an LALR(1) grammar

GATE PROBLEMS AND SOLUTIONS

Question 1

- Consider the following two statements:
- P: Every regular grammar is LL(1)
- Q: Every regular set has a LR(1) grammar
- Which of the following is TRUE? (GATE 2007)

A. Both P and Q are true
B. P is true and Q is false
C. P is false and Q is true
D. Both P and Q are false

Explanation

- A regular grammar can also be ambiguous
- For example, consider the following grammar,
- $S \rightarrow aA/a$
- $A \rightarrow aA/\varepsilon$ In above grammar, string 'a' has two leftmost derivations.
- (1) $S \rightarrow aA$
- (2) $S \rightarrow a$
- S->a (using A->ε)

- And LL(1) parses only unambiguous grammar, so statement P is False.
- Statement Q is true is for every regular set, we can have a regular grammar which is unambiguous so it can be parse by LR parser.
- So **option C** is correct choice

Question 2

A canonical set of items is given below

S → L. > R

Q → R.

On input symbol < the set has? (GATE 2014)

A. a shift-reduce conflict and a reduce-reduce conflict.
B. a shift-reduce conflict but not a reduce-reduce conflict.
C. a reduce-reduce conflict but not a shift-reduce conflict.
D. neither a shift-reduce nor a reduce-reduce conflict

Option D

Explanation

- The question is asked with respect to the symbol ' < ' which is **not present** in the given canonical set of items.
- Hence it is neither a shift-reduce conflict nor a reduce-reduce conflict on symbol '<‘.
- So **option D** is correct choice
- But if the question would have asked with respect to the symbol ' > ' then it would have been a shift-reduce conflict.

Question 3

Which of the following is true?

A. Canonical LR parser is LR (1) parser with single look ahead terminal
B. All LR(K) parsers with K > 1 can be transformed into LR(1) parsers.
C. Both (A) and (B)
D. None of the above

Option (C)

Explanation

- Canonical LR parser is LR (1) parser with single look ahead terminal. All LR(K) parsers with K > 1 can be transformed into LR(1) parsers.
- **Option (C) is correct.**

Question 4

The construction of the canonical collection of the sets of LR (1) items are similar to the construction of the canonical collection of the sets of LR (0) items. Which is an exception?

A. Closure and goto operations work a little bit different
B. Closure and goto operations work similarly
C. Closure and additive operations work a little bit different
D. Closure and associatively operations work a little bit different

- **Option (A)**

Explanation

- Closure and goto do work differently in case of LR (0) and LR (1)
- **Option (A) is correct.**

Question 5

When ß (in the LR(1) item A -> ß.a,a) is not empty, the look-head

A. Will be affecting.
B. Does not have any affect.
C. Shift will take place.
D. Reduction will take place.

Option (B)

Explanation

- There is no terminal before the non terminal beta
- **Option (B) is correct.**

Question 6

When ß is empty (A -> ß.,a), the reduction by A-> a is done

A. If next symbol is a terminal
B. Only If the next input symbol is not a
C. Only If the next input symbol is A
D. Only if the next input symbol is a

Option (D)

Explanation

- The next token is considered in this case it’s a
- **Option (D) is correct.**

Question 7

Which one from the following is false?

A. LALR parser is Bottom - Up parser
B. A parsing algorithm which performs a left to right scanning and a right most deviation is RL (1)
C. LR parser is Bottom - Up parser.
D. In LL(1), the 1 indicates that there is a one - symbol look - ahead.

option (B)

Explanation

- LALR parser is Bottom - Up parser.True
- A parsing algorithm which performs a left to right scanning and a right most deviation is RL (1).False
- LR parser is Bottom - Up parser. True
- In LL(1), the 1 indicates that there is a one - symbol look - ahead.True
- So, **option (B) is correct**.

Question 8

Which of the following is the most powerful parsing method?

A. LL(1)
B. Canonical LR
C. SLR
D. LALR

Option B

Explanation

- **Option B is correct**

Question 9

Which of the following statement is true?

A. SLR parser is more powerful than LALR.
B. LALR parser is more powerful than Canonical LR parser.
C. Canonical LR parser is more powerful than LALR parser.
D. The parsers SLR, Canonical LR, and LALR have the same power

Option C

Explanation

- **Option C is correct**

Question 10

Which of the following statements is false?

A. An unambiguous grammar has same leftmost and rightmost derivation
B. An LL(1) parser is a top-down parser
C. LALR is more powerful than SLR
D. An ambiguous grammar can never be LR(k) for any k

Option A

Explanation

- **Option A is correct**
- A grammar is ambiguous if there exists a string s such that the grammar has more than one leftmost derivations for s. We could also come up with more than one rightmost derivations for a string to prove the above proposition, but not both of right and leftmost. An unambiguous grammar can have different rightmost and leftmost derivations.

Question 11

What is the similarity between LR, LALR and SLR?

A. Use same algorithm, but different parsing table.
B. Same parsing table, but different algorithm.
C. Their Parsing tables and algorithm are similar but uses top down approach.
D. Both Parsing tables and algorithm are different.

Option A

Explanation

- The common grounds of these 3 parser is the algorithm but parsing table is different
- **Option A is correct**

Thank U