Section 3: LR Parsing

Jack Eggleston, Aaron Johnston, & Nate Yazdani

Autumn 2018

Announcements

- Scanner is due tonight
  - Be sure to test, push, and tag!
- Every person has 4 late days
  - Up to 2 can be used per assignment
  - **Submitting project components a day late uses a late day from each partner!**

Get Your LR Jargon On

- Frontier
  - The upper “layer” of the current parse tree (held in the stack)

Get Your LR Jargon On

- Frontier
  - The upper “layer” of the current parse tree (held in the stack)
- Sentential Form
  - A string that can be generated at any point in a derivation (can be reached using any number of productions from the start symbol)

Get Your LR Jargon On

- Frontier
  - The upper “layer” of the current parse tree (held in the stack)
- Sentential Form
  - A string that can be generated at any point in a derivation (can be reached using any number of productions from the start symbol)
- Handle
  - An occurrence of the right side of a production in the frontier that is used in the rightmost derivation to arrive at the current string
  - Given the derivation ... **`=> aAbcde => abbcde`**, using the production **`A => b`**:
    - The production ‘**`A => b`**’ at index 1 would be a handle of **`abbcde`**

L R (0)

Left-to-Right

Only takes one pass, performed from the left

Rightmost

At each point, finds the derivation for the rightmost handle (bottom-up)

No Lookahead

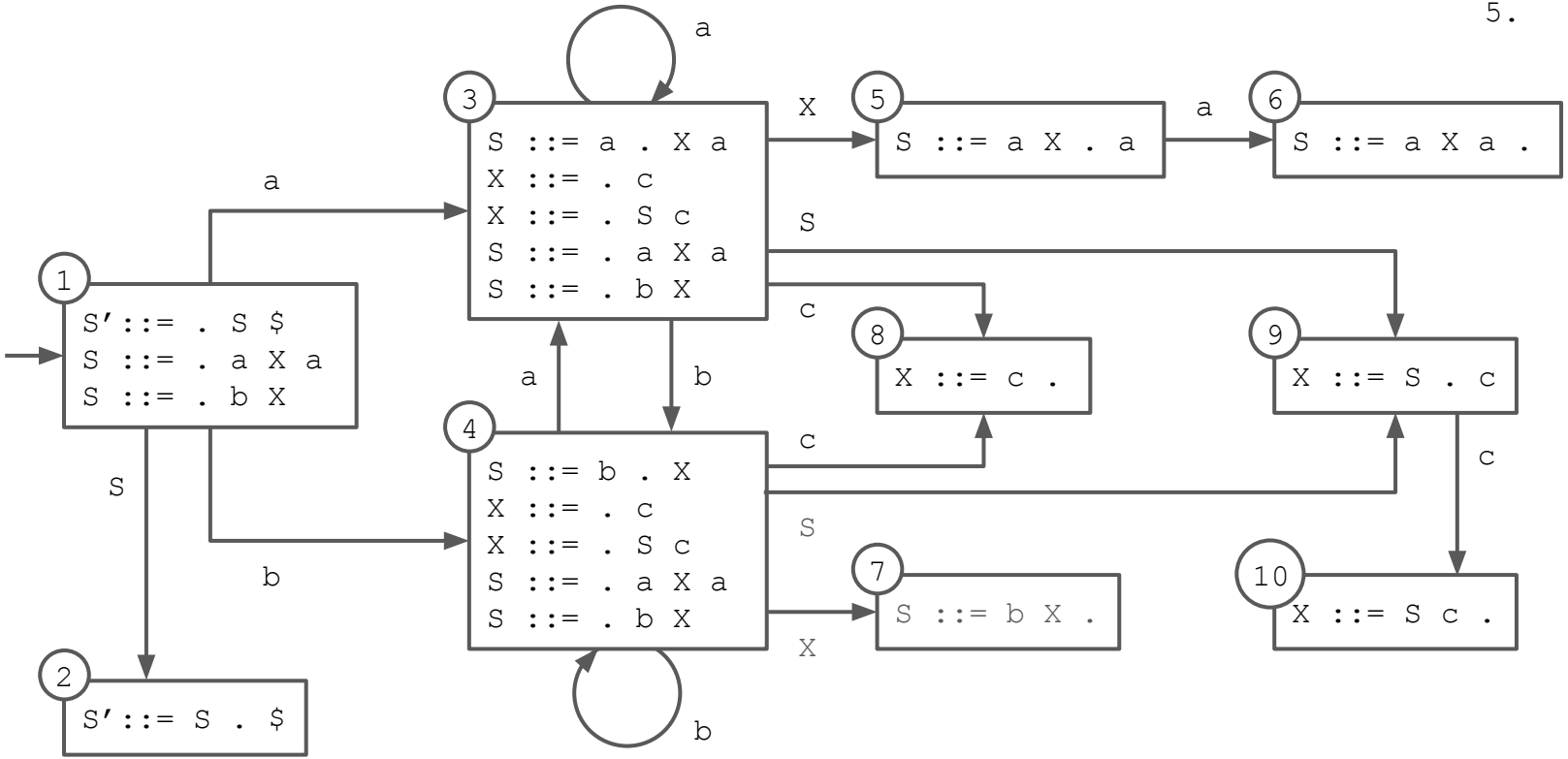
After shifting a single token from the input, has enough information to proceed

Problem 1 (On Worksheet)

1. S′ ::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c

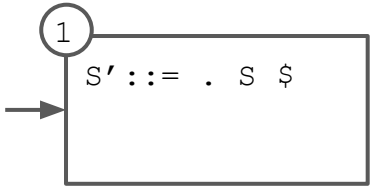
1. S′ ::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c

Completed State Diagram



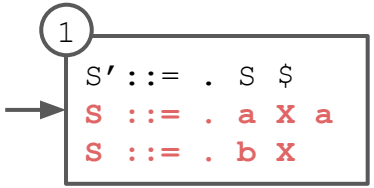
State Diagram Construction

1. S’::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



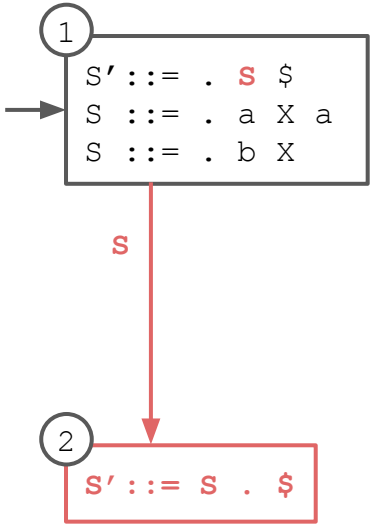
State Diagram Construction

1. S’::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



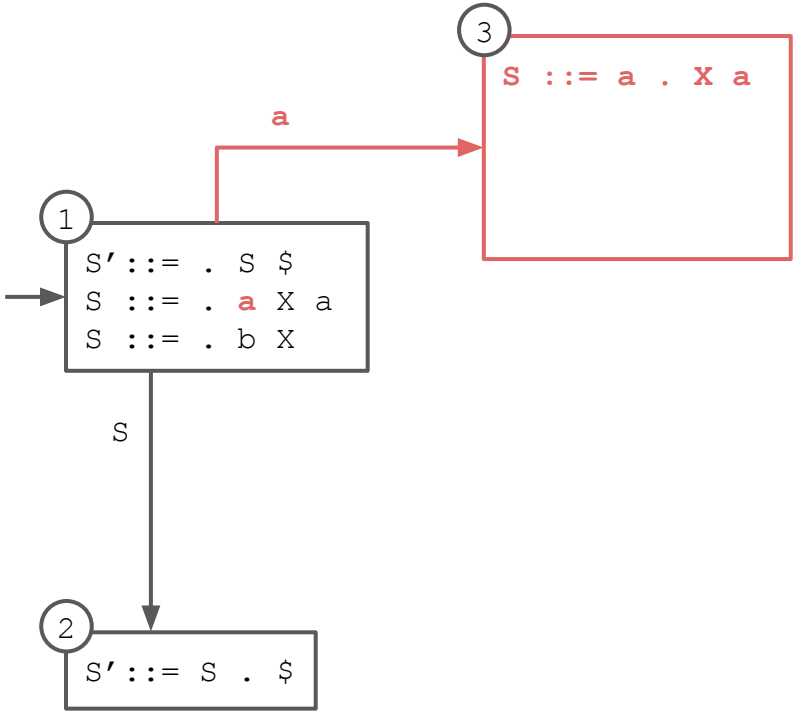
State Diagram Construction

1. S’::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



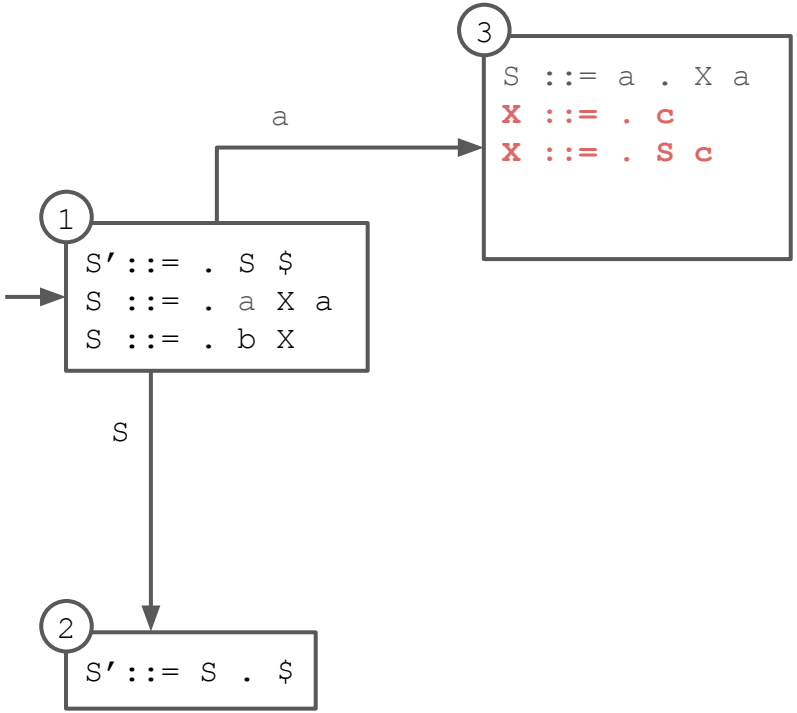
State Diagram Construction

1. S’::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



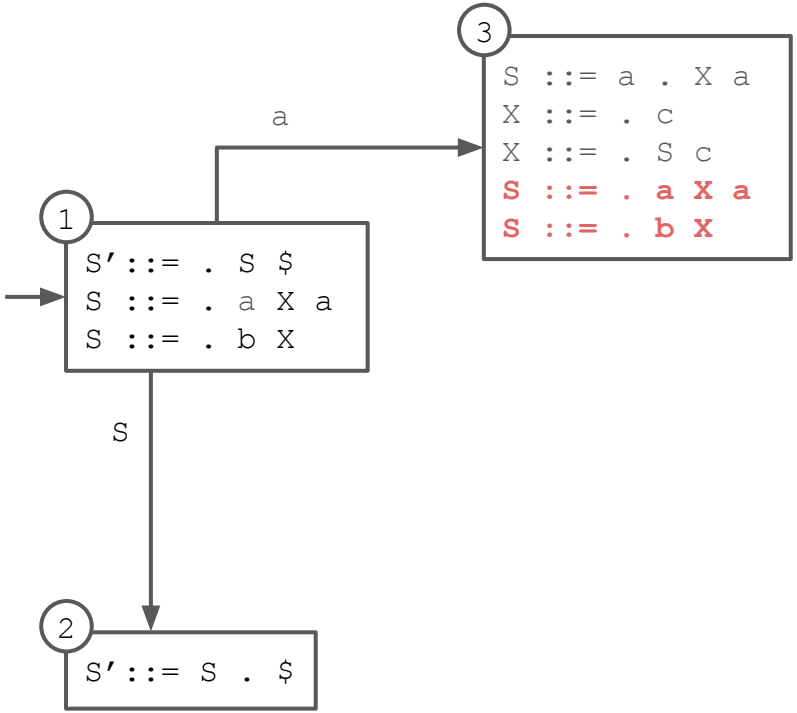
State Diagram Construction

1. S′ ::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



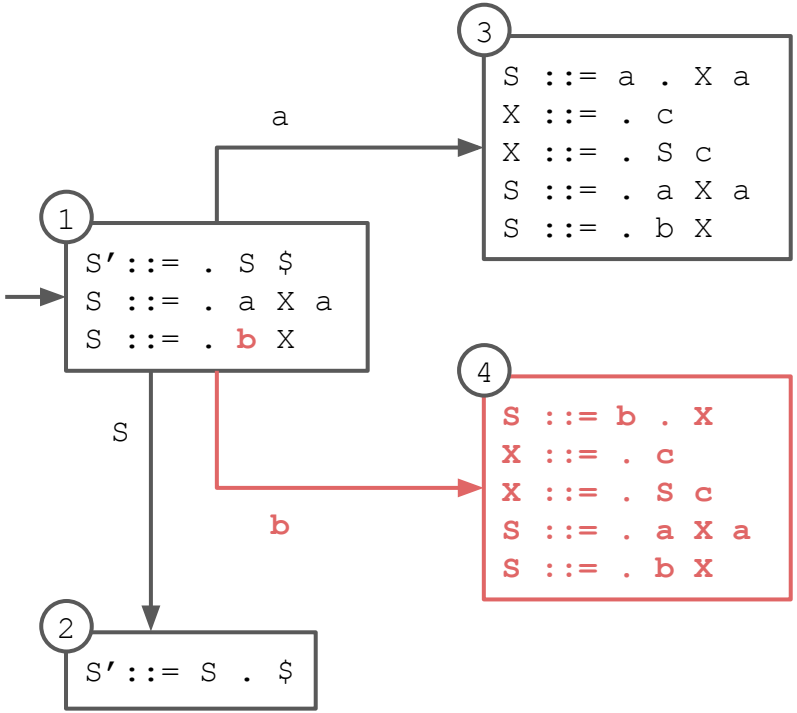
State Diagram Construction

1. S’::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



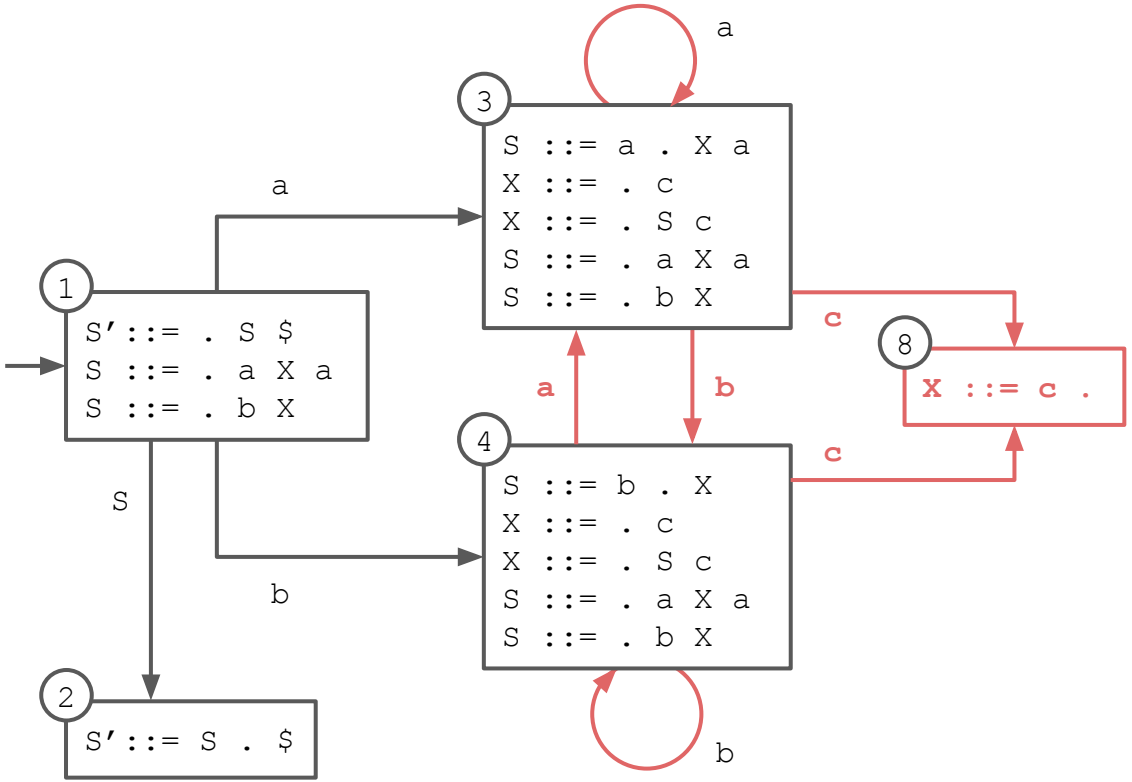
State Diagram Construction

1. S′ ::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



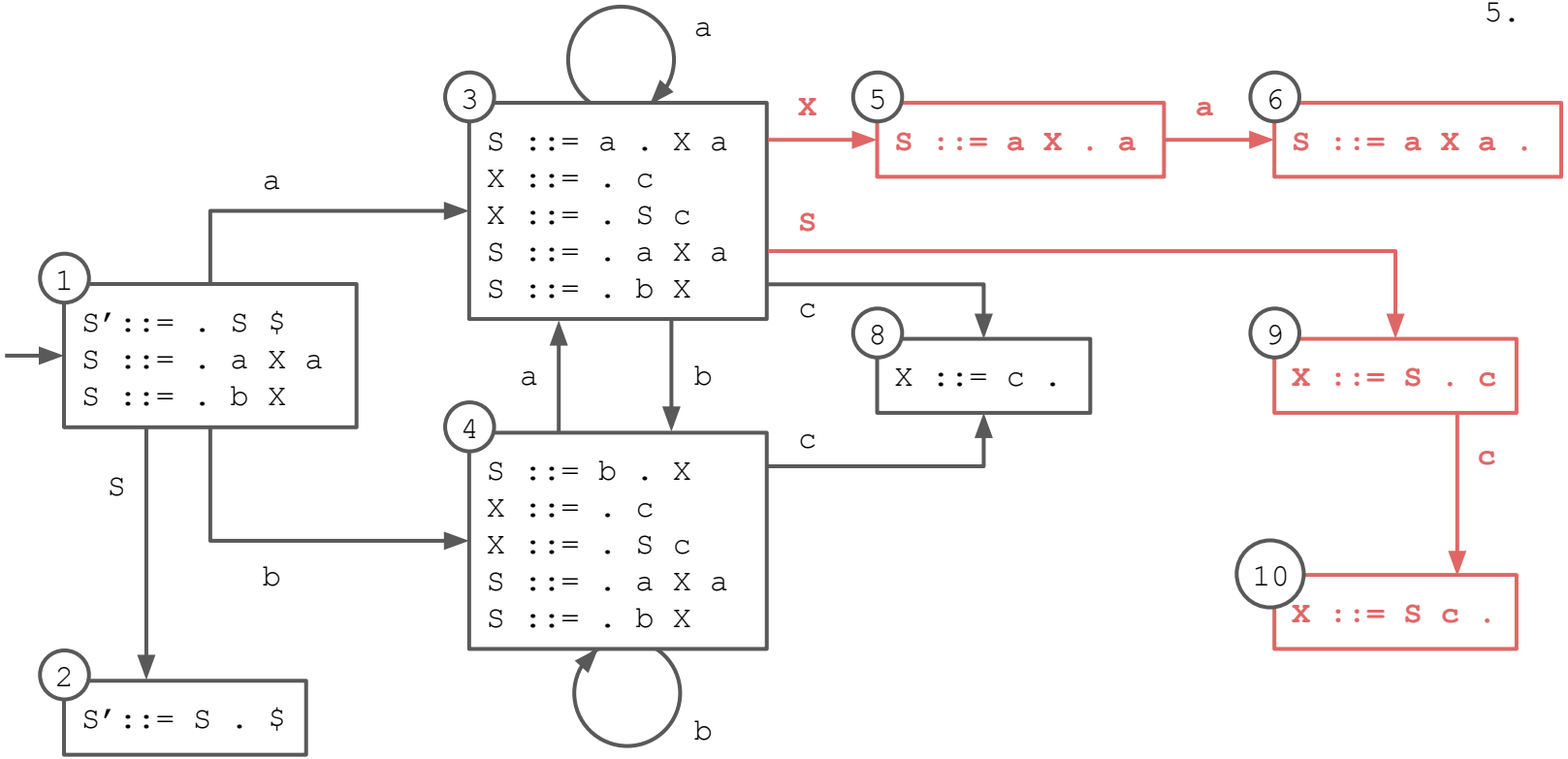
State Diagram Construction

1. S′ ::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



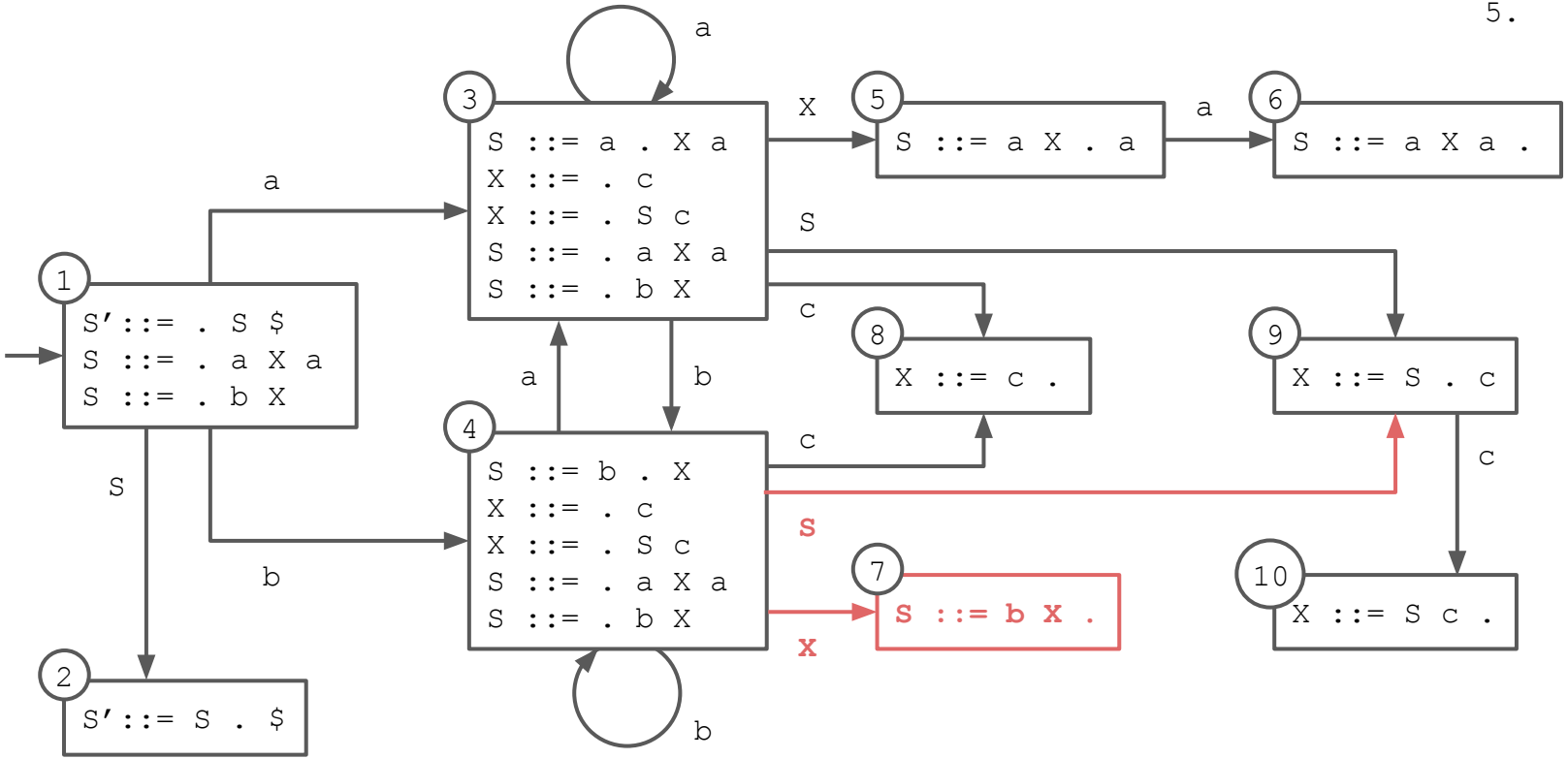
State Diagram Construction

1. S′ ::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



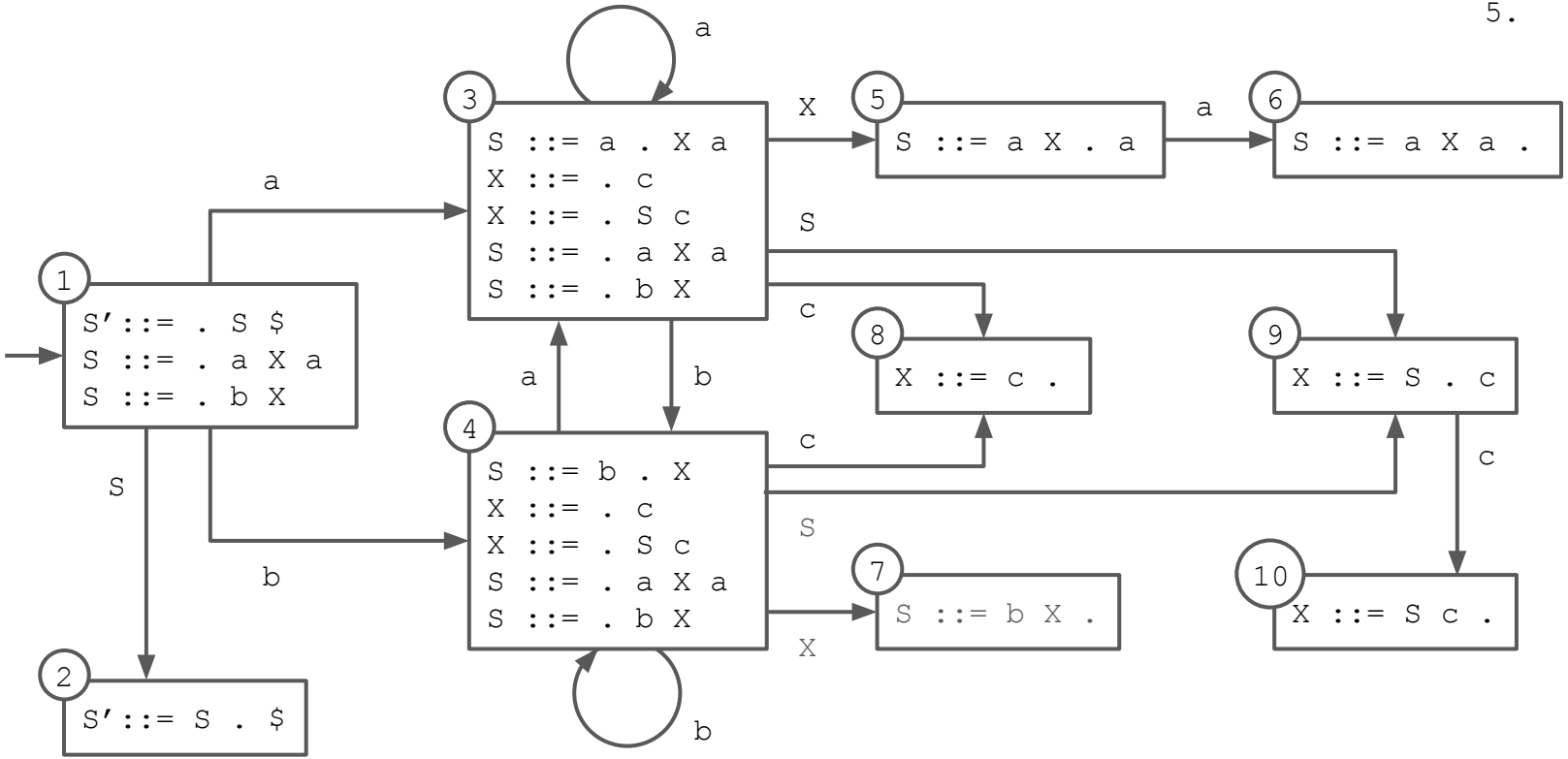
State Diagram Construction

1. S′ ::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c



1. S′ ::= S $
2. S ::= a X a
3. S ::= b X
4. X ::= c
5. X ::= S c

State Diagram Construction



Converted to Table

`s#` means "shift and enter state #"
- occurs when there is a transition on a terminal

`r#` means "reduce using production #"
- occurs when a state contains an item with the location at the end of the right-hand side

`g#` means "go to state #"
- occurs when there is a transition on a nonterminal

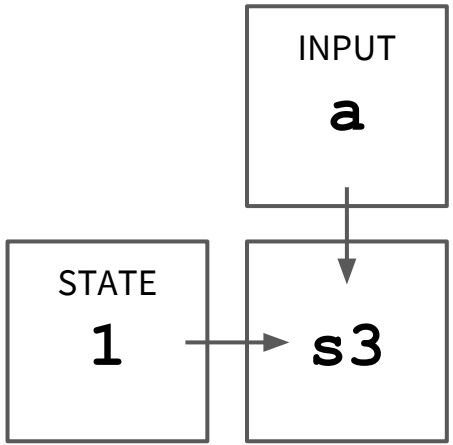
`acc` means "accept"
- occurs when the start symbol (S here) has been completed and there is no more input

STATE	ACTION				GOTO	
	a	b	c	$	S	X
1	s3	s4			g2	
2				acc		
3	s3	s4	s8		g9	g5
4	s3	s4	s8		g9	g7
5	s6					
6	r2	r2	r2	r2		
7	r3	r3	r3	r3		
8	r4	r4	r4	r4		
9			s10			
10	r5	r5	r5	r5		

Parse Trace

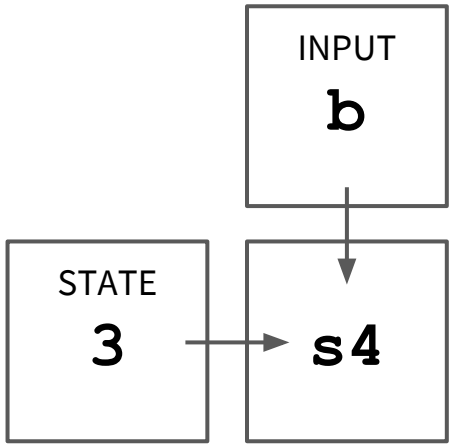
STACK	INPUT
$ s_1	a b c c a $

Parse Trace



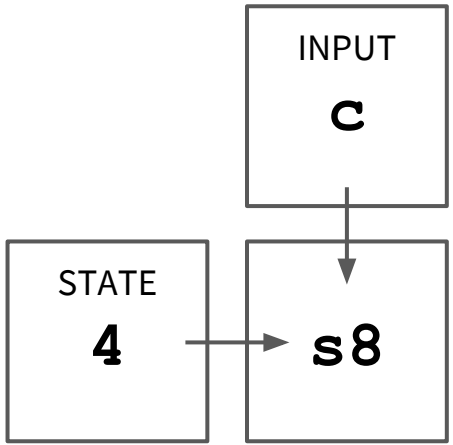
STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $

Parse Trace



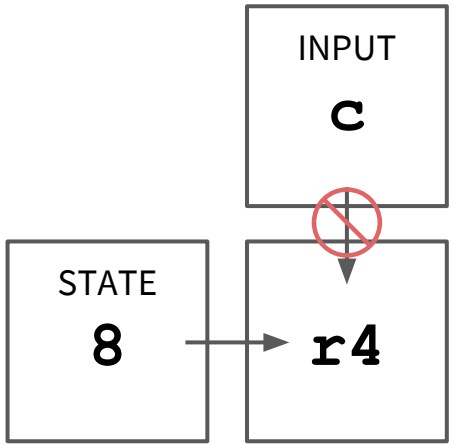
STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $

Parse Trace



STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $

Parse Trace

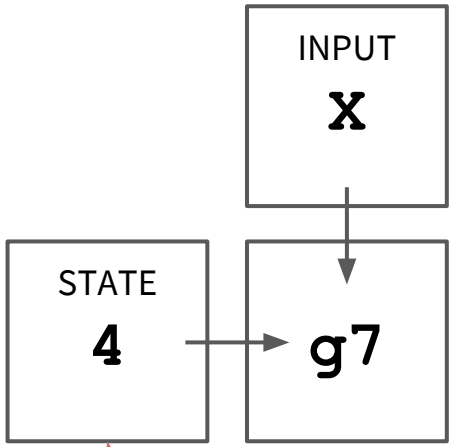


4. X ::= c

STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X	c a $

For LR(0), the input being c doesn't matter here: state 8 always reduces by production 4 regardless of the next input

Parse Trace

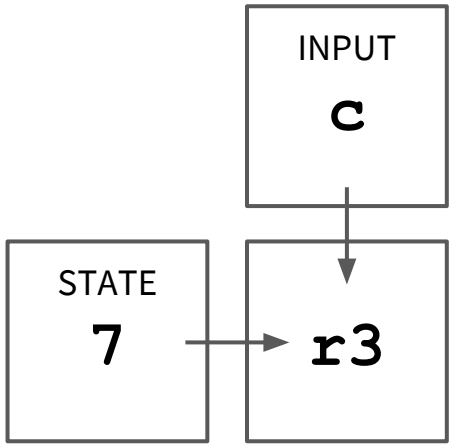


STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $

After a reduction, we go back to the last state on the stack and use the reduced non-terminal to determine what state to GOTO.

This allows the parser to run in O(n) time, since it doesn't have to re-evaluate the entire stack!

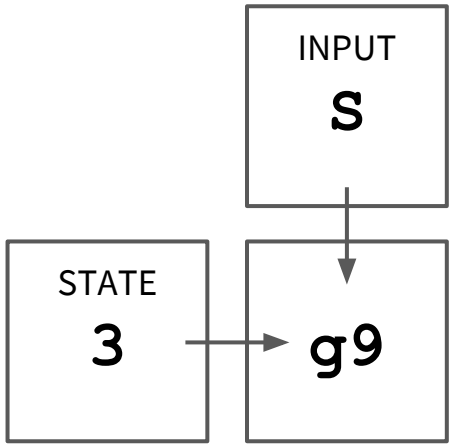
Parse Trace



STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S	c a $

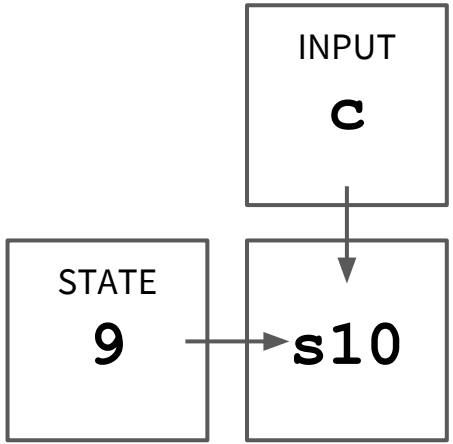
3. S ::= b X

Parse Trace



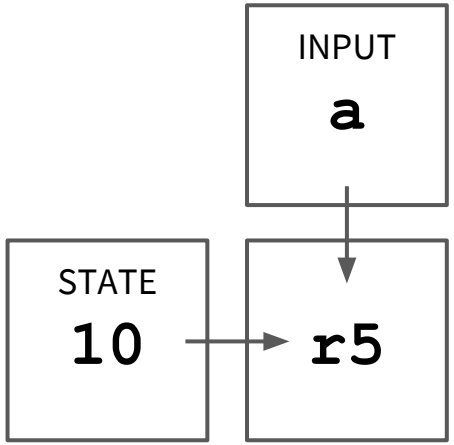
STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S s_9	c a $

Parse Trace



STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S s_9	c a $
$ s_1 a s_3 S s_9 c s_{10}	**a $**

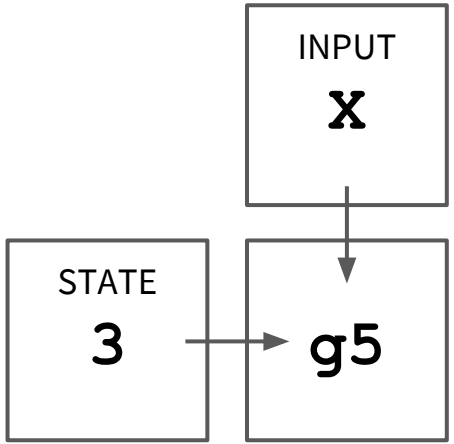
Parse Trace



STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S s_9	c a $
$ s_1 a s_3 S s_9 c s_{10}	a $
$ s_1 a s_3 X	a $

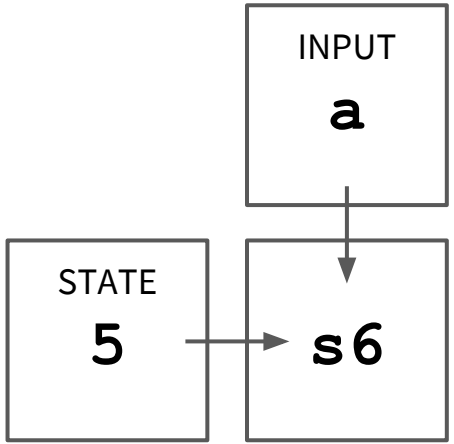
5. X ::= S c

Parse Trace



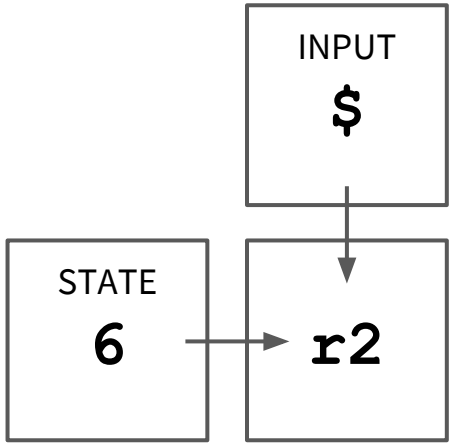
STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S s_9	c a $
$ s_1 a s_3 S s_9 c s_{10}	a $
$ s_1 a s_3 X **s_5**	a $

Parse Trace



STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S s_9	c a $
$ s_1 a s_3 S s_9 c s_{10}	a $
$ s_1 a s_3 X s_5	a $
$ s_1 a s_3 X s_5 a s_6	**$**

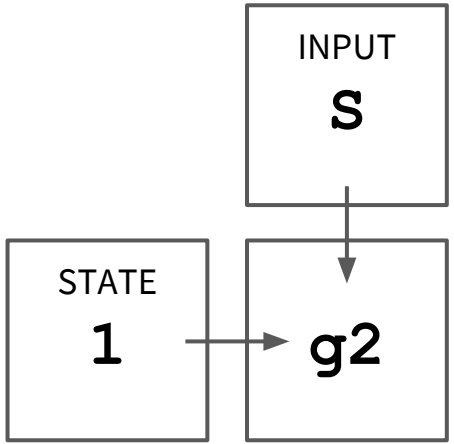
Parse Trace



STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S s_9	c a $
$ s_1 a s_3 S s_9 c s_{10}	a $
$ s_1 a s_3 X s_5	a $
$ s_1 a s_3 X s_5 a s_6	$
$ s_1 S	$

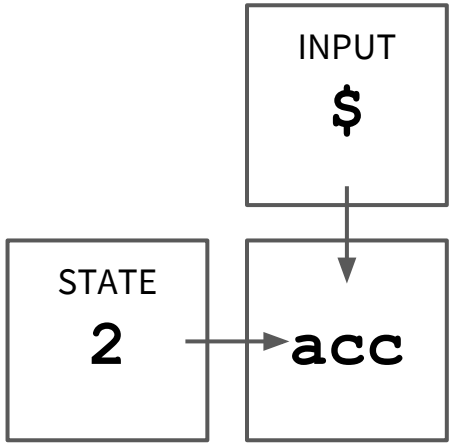
2. S ::= a X a

Parse Trace



STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S s_9	c a $
$ s_1 a s_3 S s_9 c s_{10}	a $
$ s_1 a s_3 X s_5	a $
$ s_1 a s_3 X s_5 a s_6	$
$ s_1 S s_2	$

Parse Trace



STACK	INPUT
$ s_1	a b c c a $
$ s_1 a s_3	b c c a $
$ s_1 a s_3 b s_4	c c a $
$ s_1 a s_3 b s_4 c s_8	c a $
$ s_1 a s_3 b s_4 X s_7	c a $
$ s_1 a s_3 S s_9	c a $
$ s_1 a s_3 S s_9 c s_{10}	a $
$ s_1 a s_3 X s_5	a $
$ s_1 a s_3 X s_5 a s_6	$
$ s_1 S s_2	$
accept!	$