Section 3: LR Parsing

Kory Watson, Aaron Johnston, Miya Natsuhara, Sam Wolfson

Autumn 2019

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
  - Each late day gives an extra 24 hour chunk (including weekend days)
  - **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

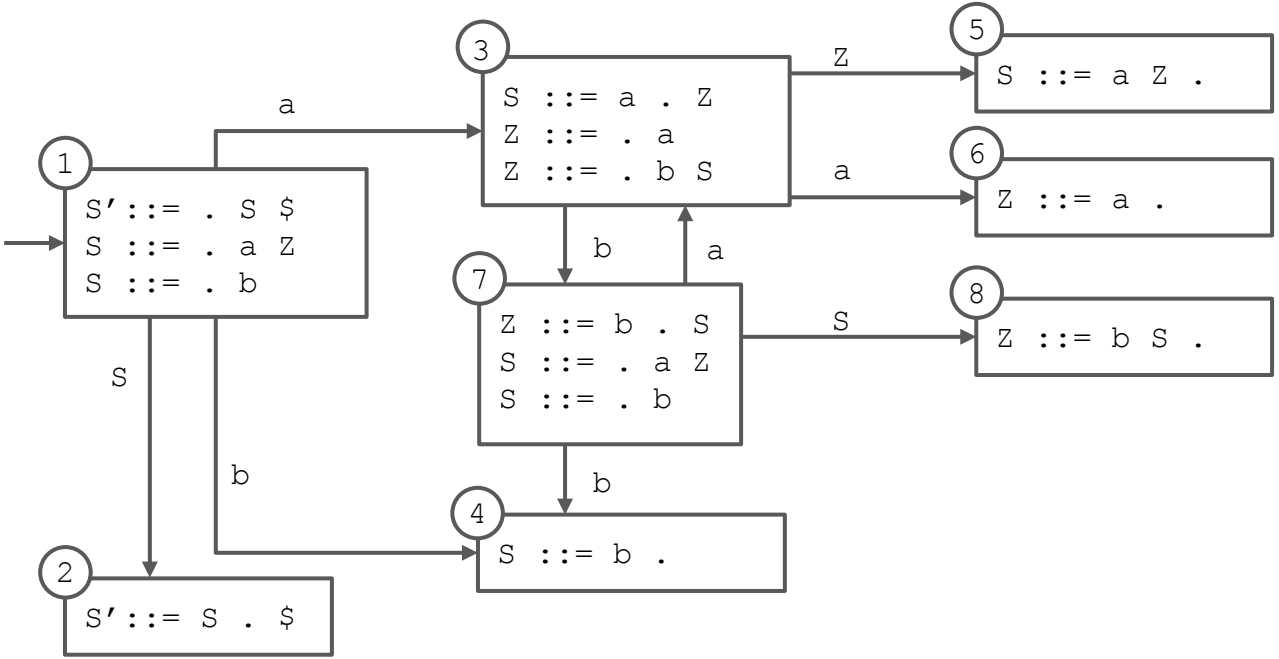
After shifting a single token from the input, has enough information to proceed

Problem 1 (On Worksheet)

```
0. S′ ::= S $
1. S  ::= a Z
2. S  ::= b
3. Z  ::= a
4. Z  ::= b S
```

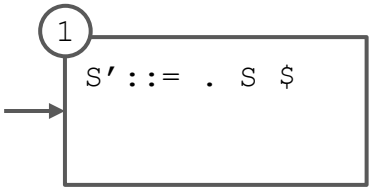
Completed State Diagram

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



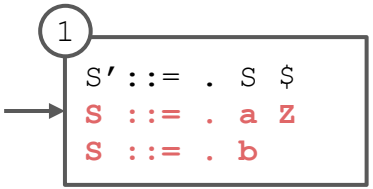
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



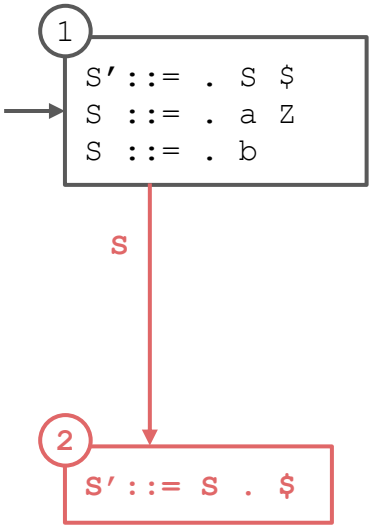
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



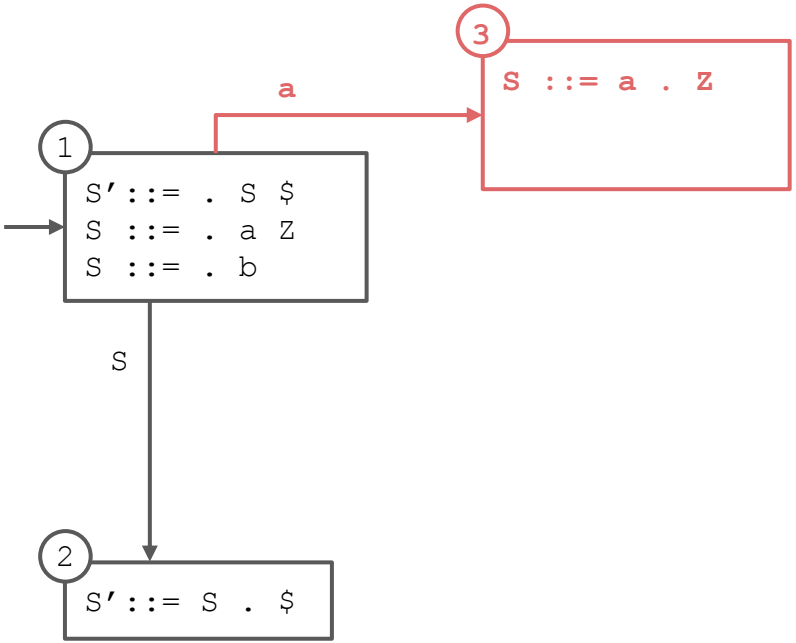
State Diagram Construction

```
0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S
```



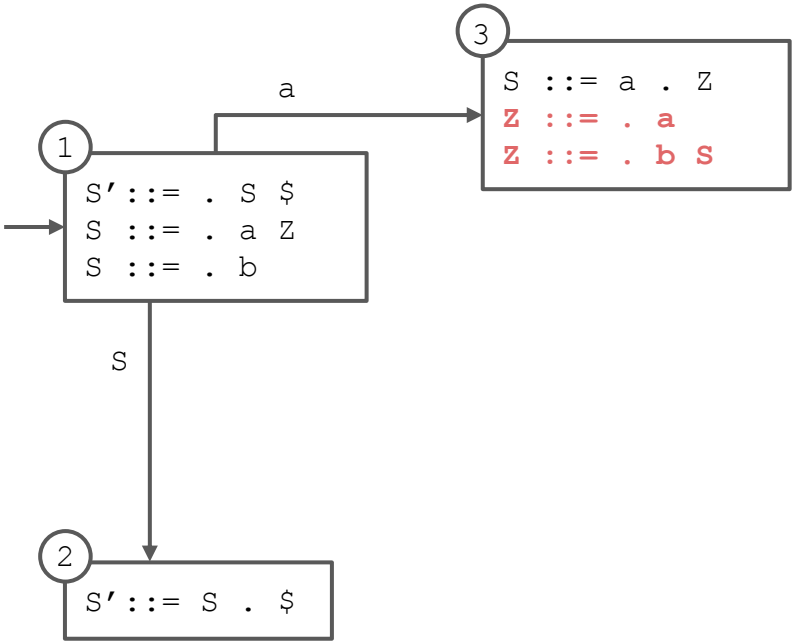
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



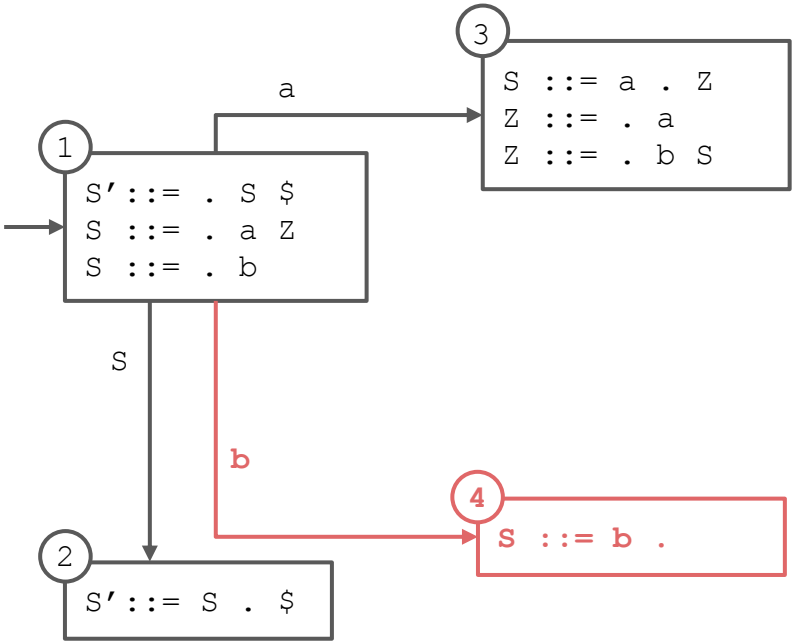
State Diagram Construction

```
0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S
```



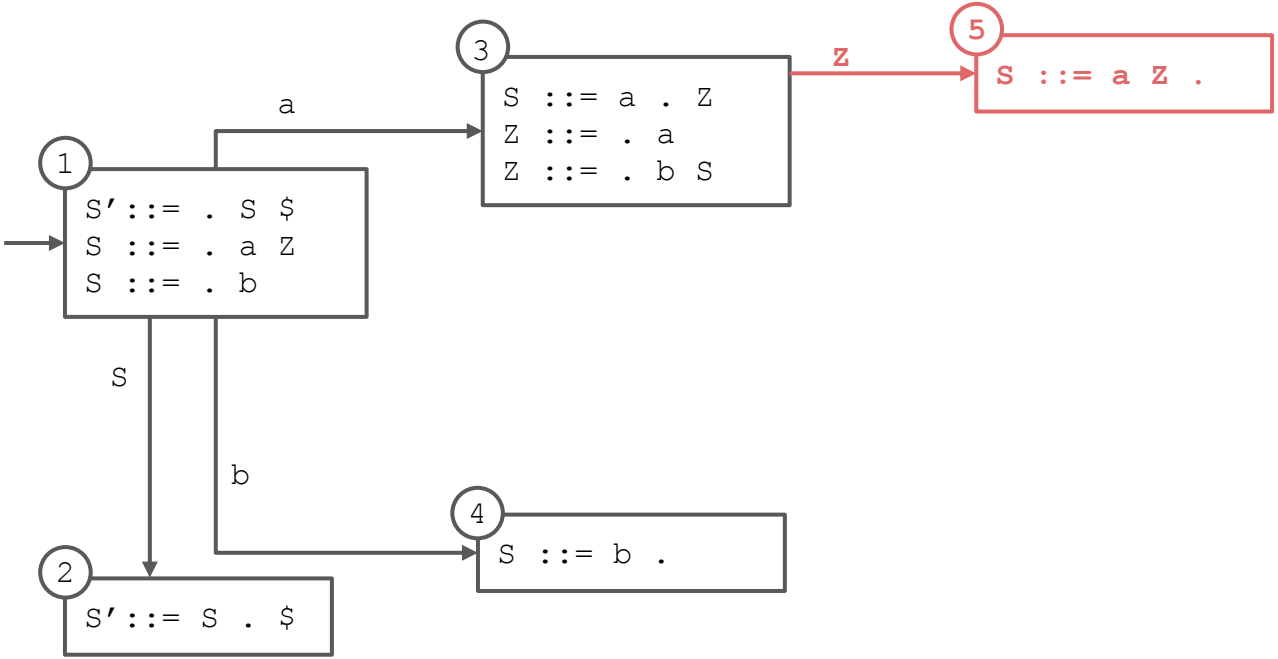
State Diagram Construction

```
0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S
```



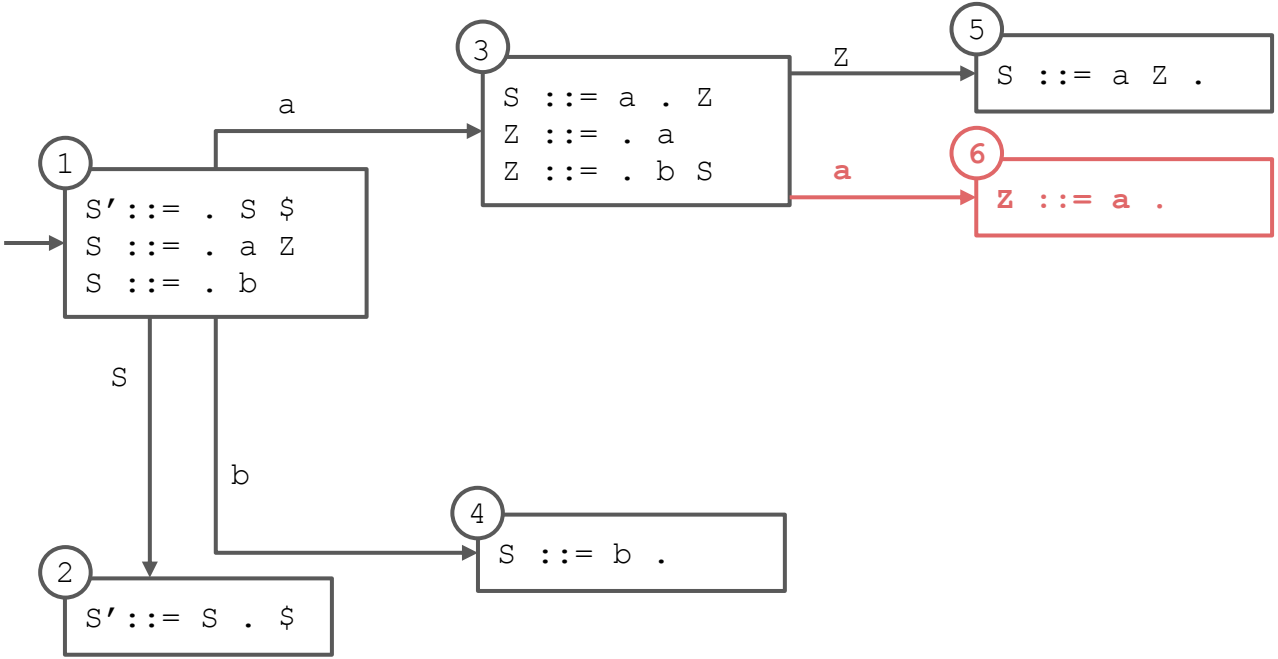
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



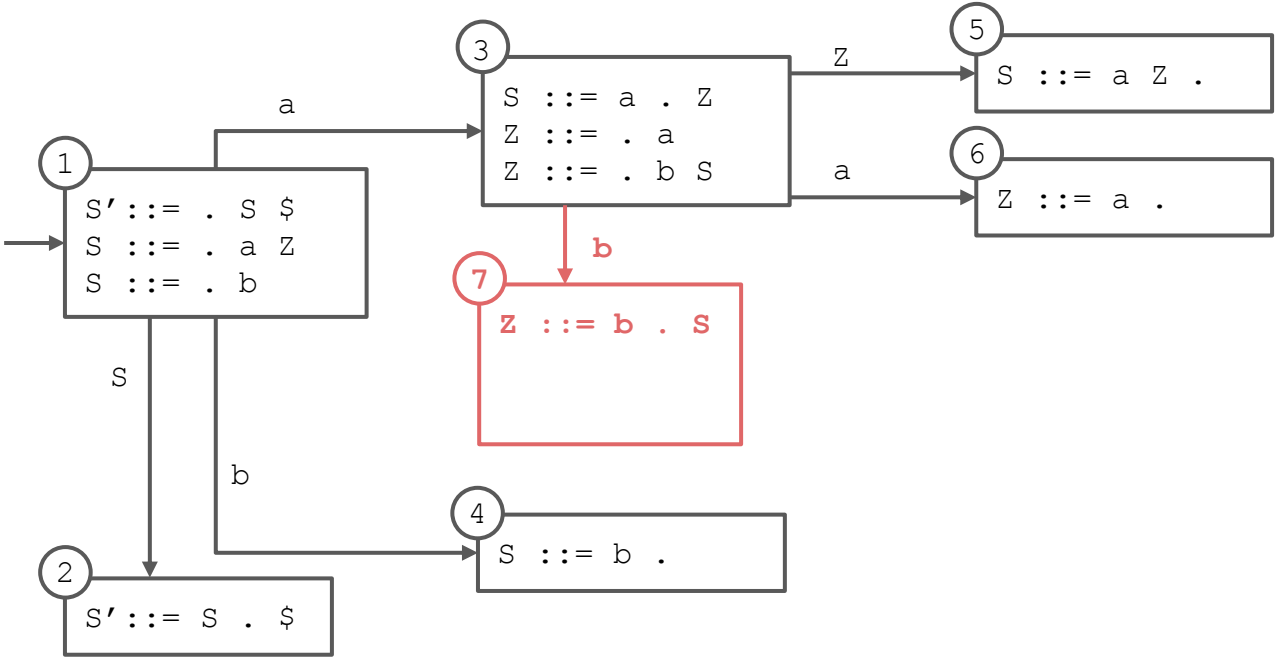
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



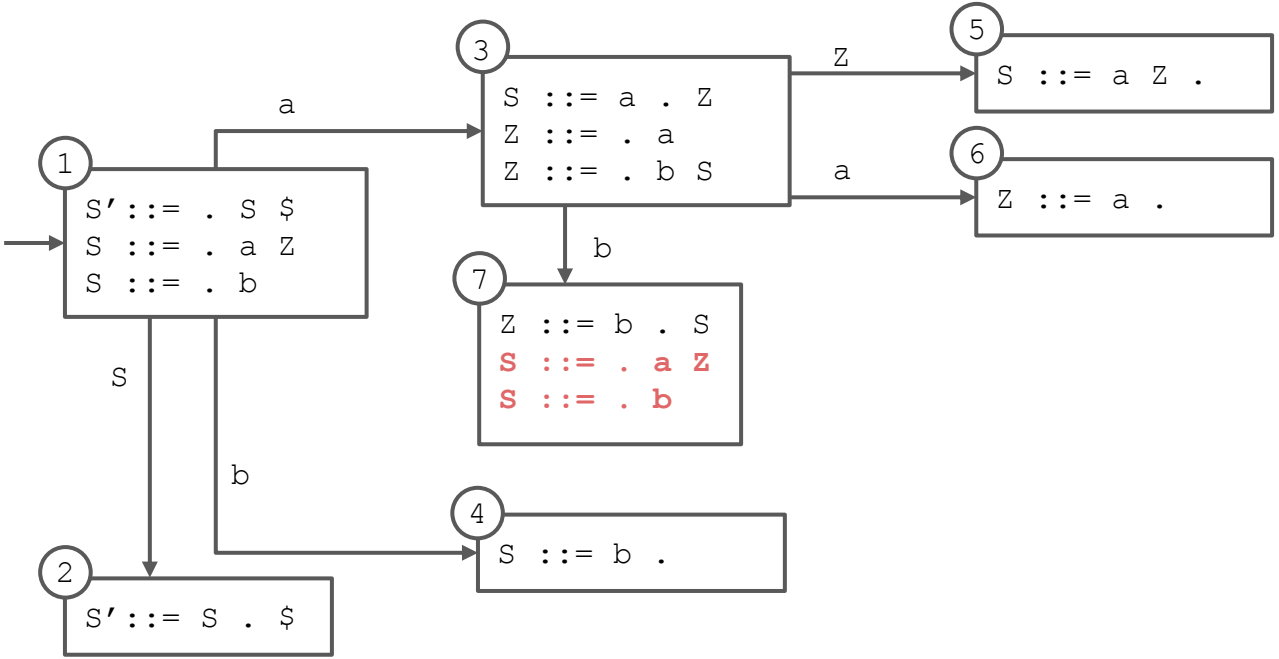
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



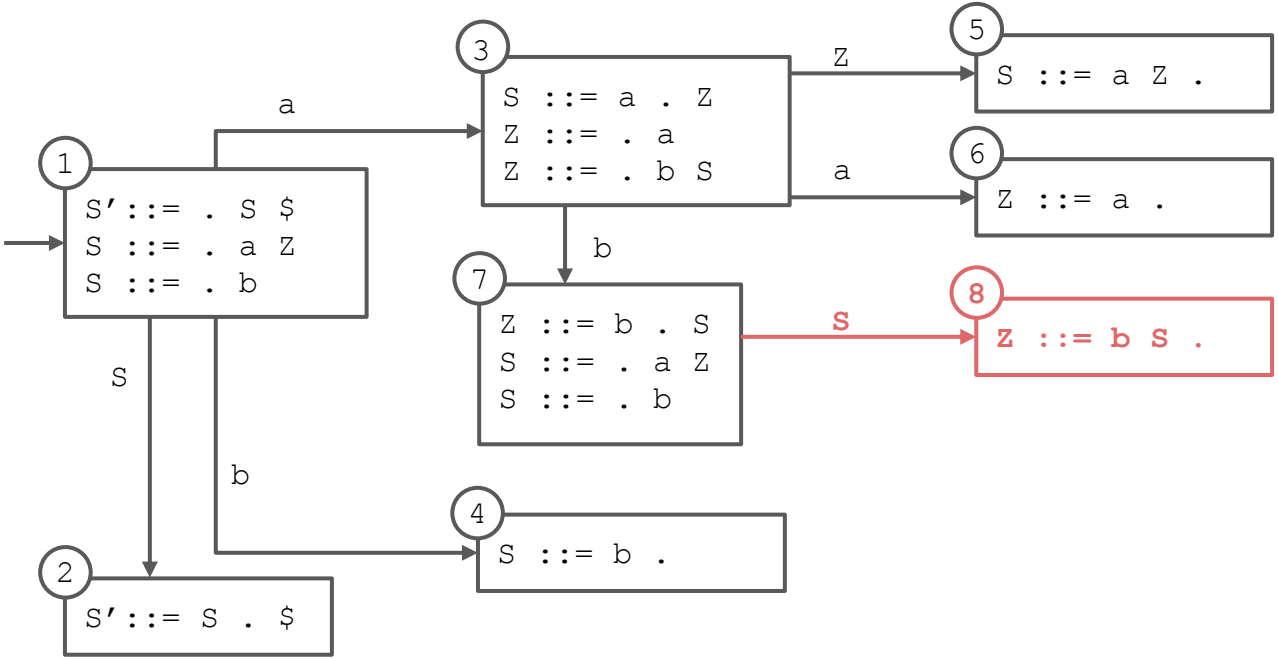
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



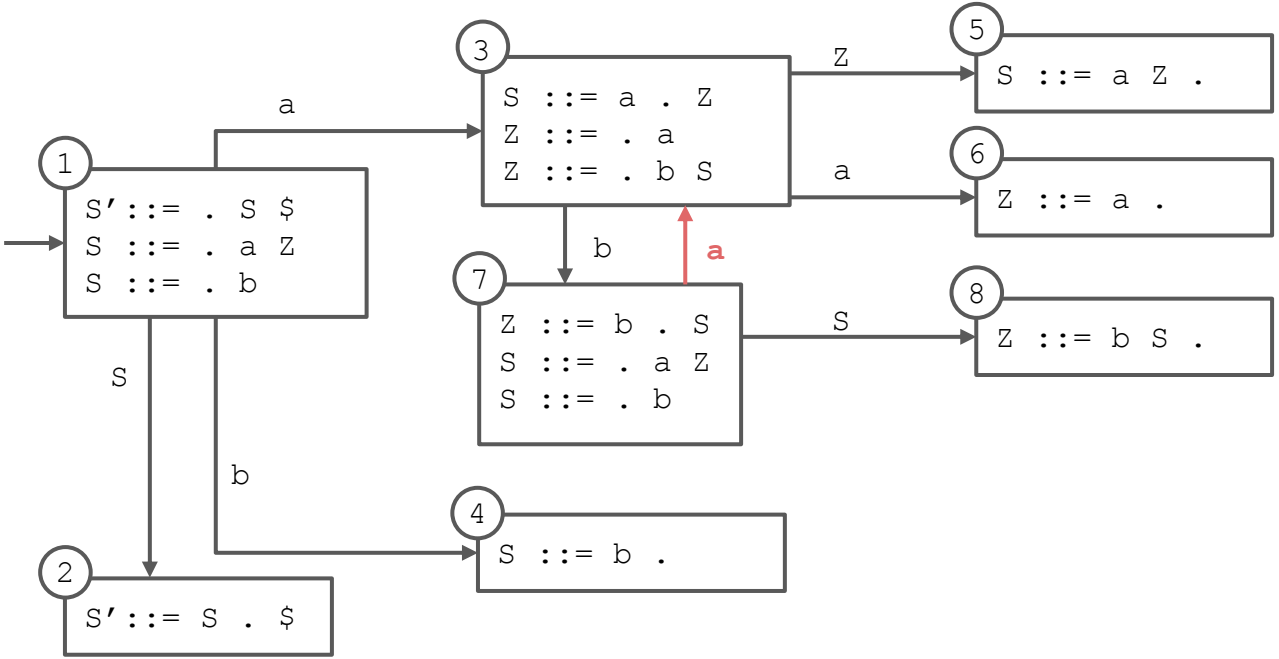
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



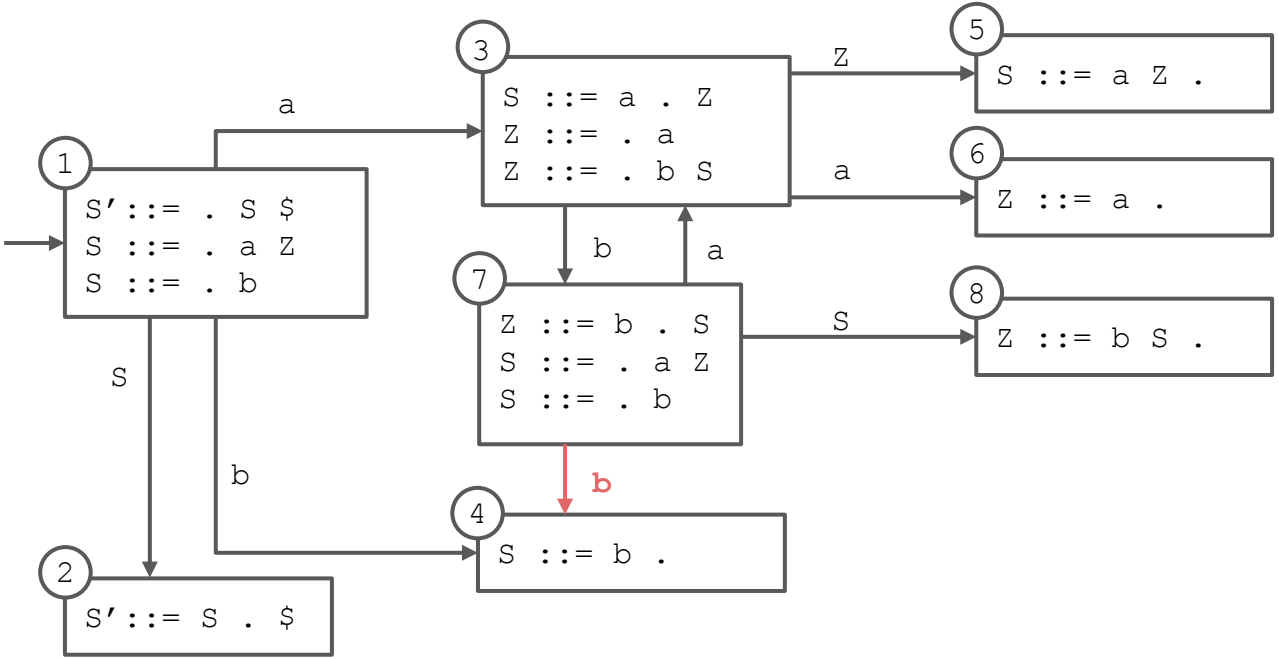
State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



State Diagram Construction

0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S



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

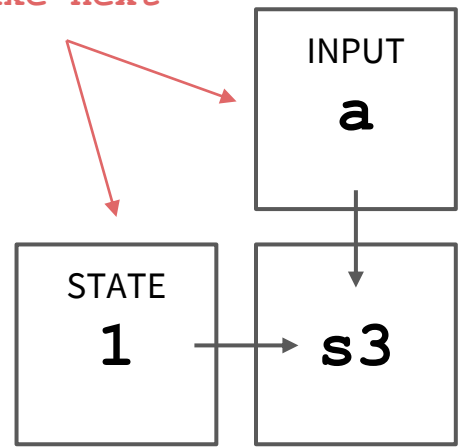
STATE	ACTION			GOTO	
	a	b	$	S	Z
1	s3	s4		g2	
2			acc		
3	s6	s7			g5
4	r2	r2	r2		
5	r1	r1	r1		
6	r3	r3	r3		
7	s3	s4		g8	
8	r4	r4	r4		

Parse Trace

STACK	INPUT	ACTION
$ s1	a b a b b $	

Parse Trace

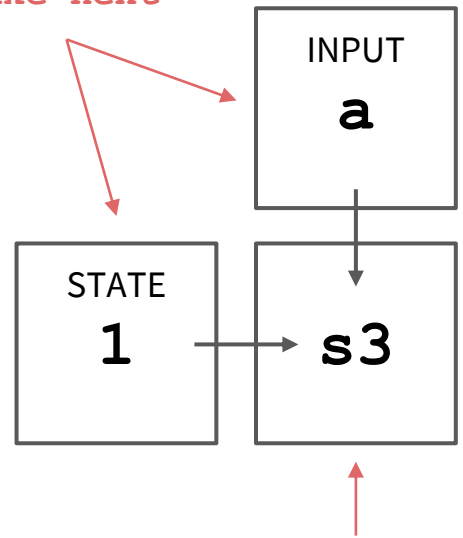
Row and column of table to look up: decides what action to take next



STACK	INPUT	ACTION
$ s1 $ s1 a	a b a b b $ b a b b $	SHIFT

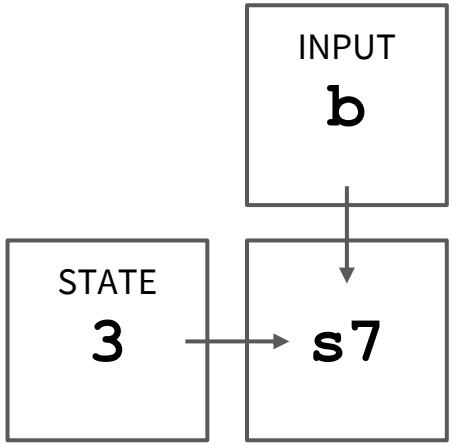
Parse Trace

Row and column of table to look up: decides what action to take next



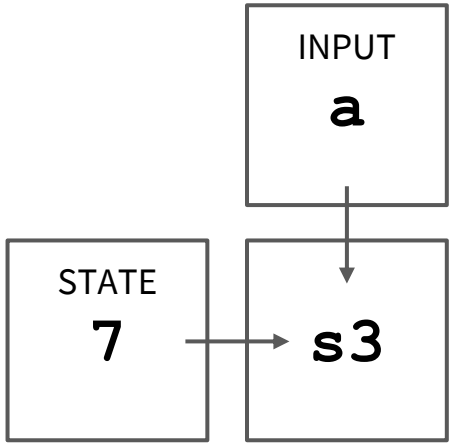
STACK	INPUT	ACTION
$ s1 $ s1 a **s3**	**a b a b b $** b a b b $	SHIFT

Parse Trace



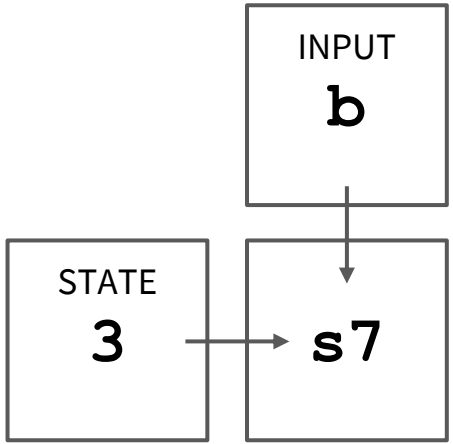
STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	**SHIFT**
$ s1 a s3 b s7	**a b b $**	

Parse Trace



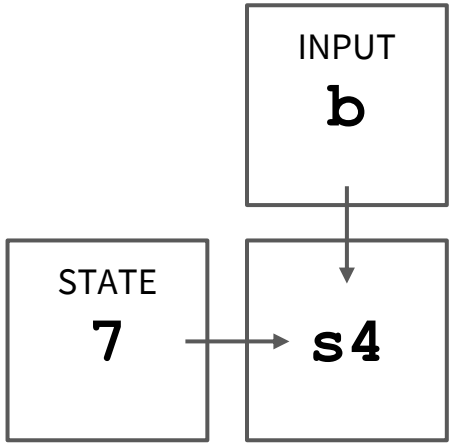
STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	**SHIFT**
$ s1 a s3 b s7 a s3	**b b $**	

Parse Trace



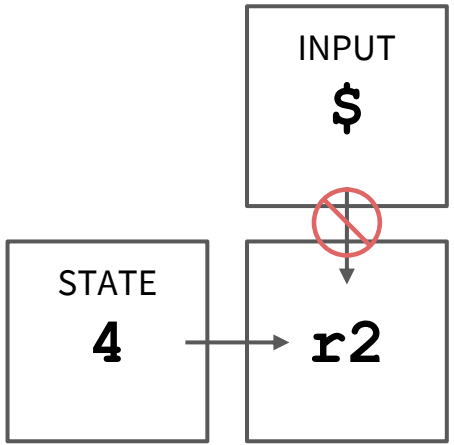
STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	**SHIFT**
$ s1 a s3 b s7 a s3 b s7	**b $**	

Parse Trace



STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	SHIFT
$ s1 a s3 b s7 a s3 b s7	b $	**SHIFT**
$ s1 a s3 b s7 a s3 b s7 b s4	**$**	

Parse Trace

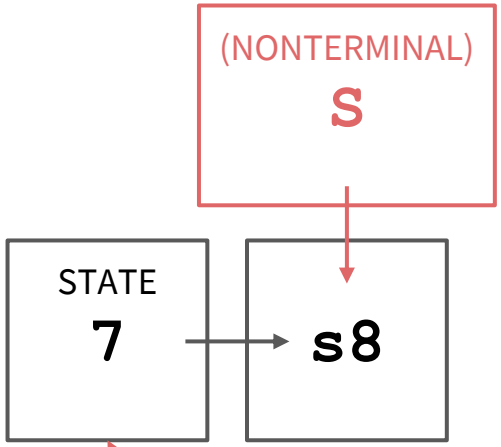


2. S ::= b

STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	SHIFT
$ s1 a s3 b s7 a s3 b s7	b $	SHIFT
$ s1 a s3 b s7 a s3 b s7 b s4	$	REDUCE
$ s1 a s3 b s7 a s3 b s7 S	**$**	

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

Parse Trace

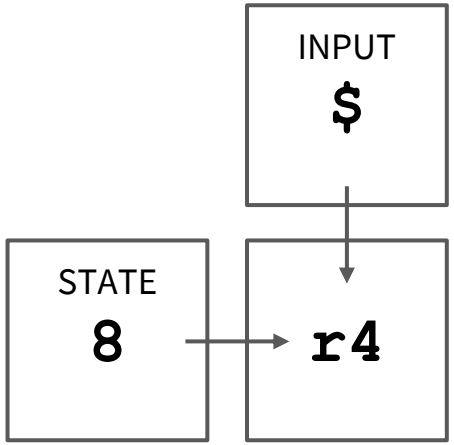


STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	SHIFT
$ s1 a s3 b s7 a s3 b s7	b $	SHIFT
$ s1 a s3 b s7 a s3 b s7 b s4	$	REDUCE
$ s1 a s3 b s7 a s3 b s7 S **s8**	$	

After a reduction, we go back to a previous 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

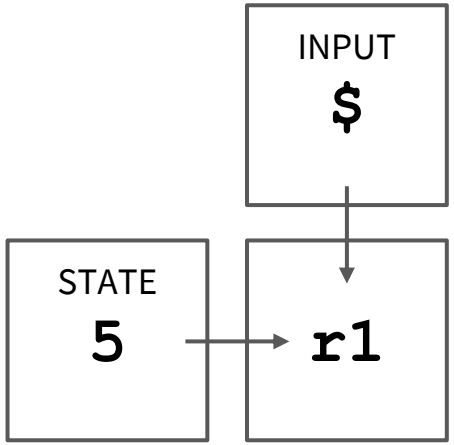


4. Z ::= b S

(and GOTO step:
s3 & Z → g5)

STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	SHIFT
$ s1 a s3 b s7 a s3 b s7	b $	SHIFT
$ s1 a s3 b s7 a s3 b s7 b s4	$	REDUCE
$ s1 a s3 b s7 a s3 b s7 S s8	$	**REDUCE**
$ s1 a s3 b s7 a s3 Z s5	**$**	

Parse Trace

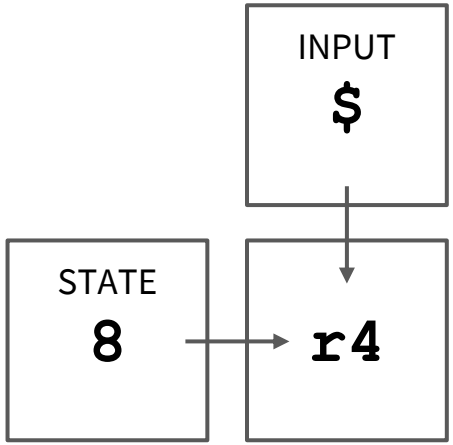


1. S ::= a Z

(and GOTO step:
s7 & S → g8)

STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	SHIFT
$ s1 a s3 b s7 a s3 b s7	b $	SHIFT
$ s1 a s3 b s7 a s3 b s7 b s4	$	REDUCE
$ s1 a s3 b s7 a s3 b s7 S s8	$	REDUCE
$ s1 a s3 b s7 a s3 Z s5	$	**REDUCE**
$ s1 a s3 b s7 S s8	**$**	

Parse Trace

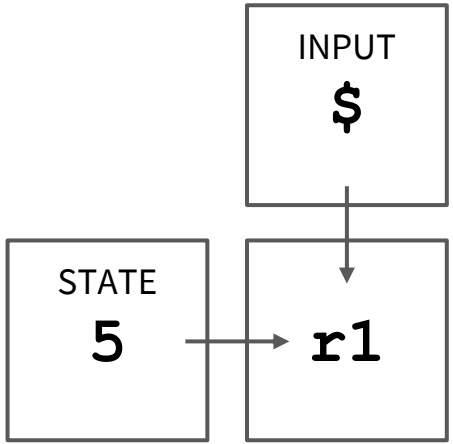


4. Z ::= b S

(and GOTO step:
s3 & Z → g5)

STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	SHIFT
$ s1 a s3 b s7 a s3 b s7	b $	SHIFT
$ s1 a s3 b s7 a s3 b s7 b s4	$	REDUCE
$ s1 a s3 b s7 a s3 b s7 S s8	$	REDUCE
$ s1 a s3 b s7 a s3 Z s5	$	REDUCE
$ s1 a s3 b s7 S s8	$	**REDUCE**
$ s1 a s3 Z s5	**$**	

Parse Trace

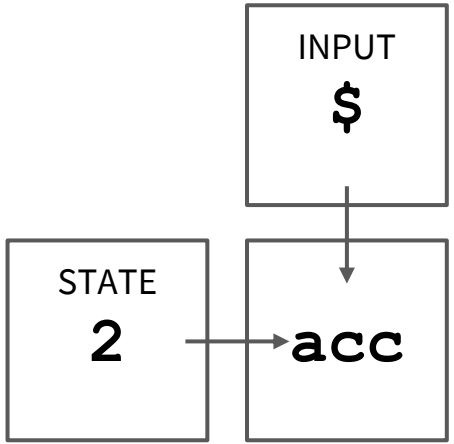


1. S ::= a Z

(and GOTO step: s1 & S → g2)

STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	SHIFT
$ s1 a s3 b s7 a s3 b s7	b $	SHIFT
$ s1 a s3 b s7 a s3 b s7 b s4	$	REDUCE
$ s1 a s3 b s7 a s3 b s7 S s8	$	REDUCE
$ s1 a s3 b s7 a s3 Z s5	$	REDUCE
$ s1 a s3 b s7 S s8	$	REDUCE
$ s1 a s3 Z s5	$	**REDUCE**
$ s1 S s2	**$**	

Parse Trace



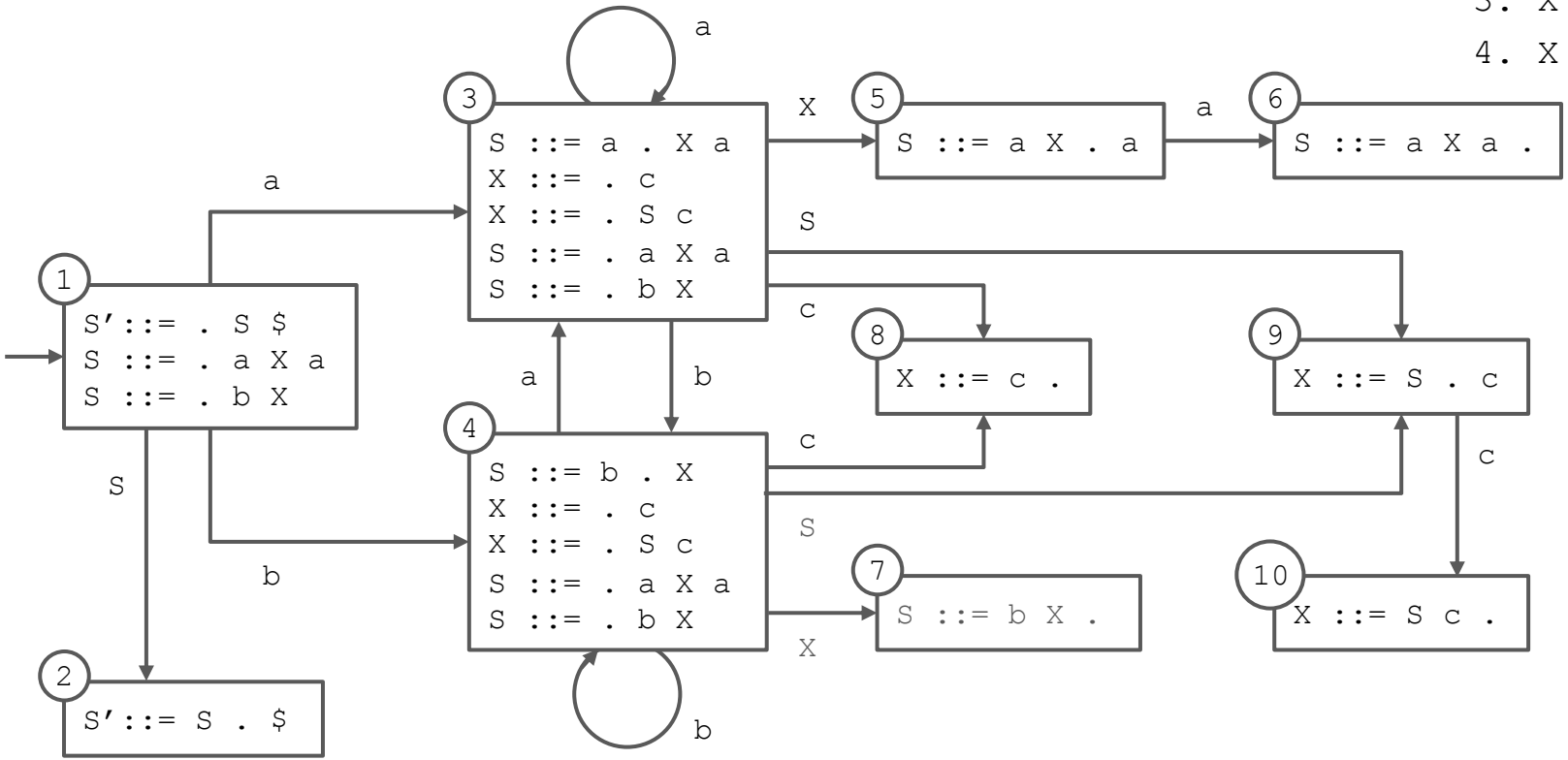
STACK	INPUT	ACTION
$ s1	**a b a b b $**	SHIFT
$ s1 a s3	b a b b $	SHIFT
$ s1 a s3 b s7	a b b $	SHIFT
$ s1 a s3 b s7 a s3	b b $	SHIFT
$ s1 a s3 b s7 a s3 b s7	b $	SHIFT
$ s1 a s3 b s7 a s3 b s7 b s4	$	REDUCE
$ s1 a s3 b s7 a s3 b s7 S s8	$	REDUCE
$ s1 a s3 b s7 a s3 Z s5	$	REDUCE
$ s1 a s3 b s7 S s8	$	REDUCE
$ s1 a s3 Z s5	$	REDUCE
$ s1 S s2	$	**ACCEPT**

Problem 2 (On Worksheet)

```
0. S' ::= S $
1. S  ::= a X a
2. S  ::= b X
3. X  ::= c
4. X  ::= S c
```

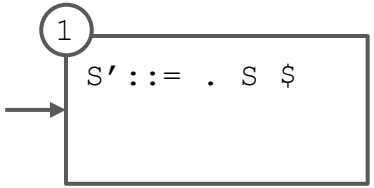
Completed State Diagram

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



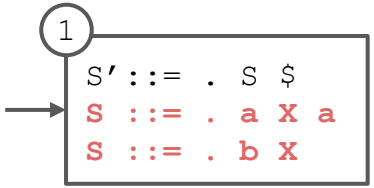
State Diagram Construction

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



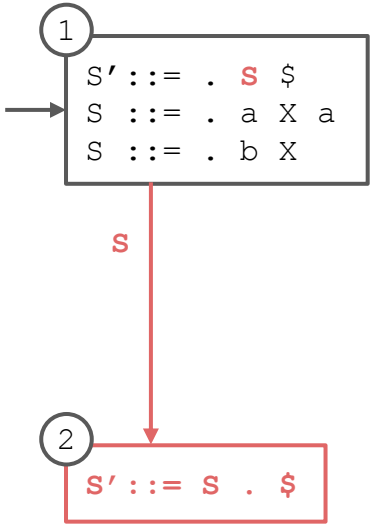
State Diagram Construction

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



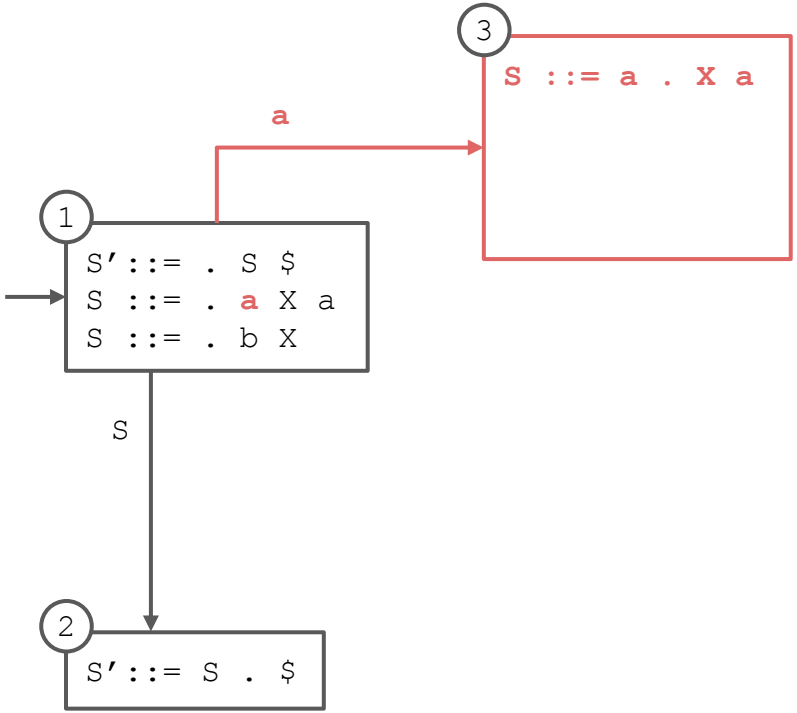
State Diagram Construction

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



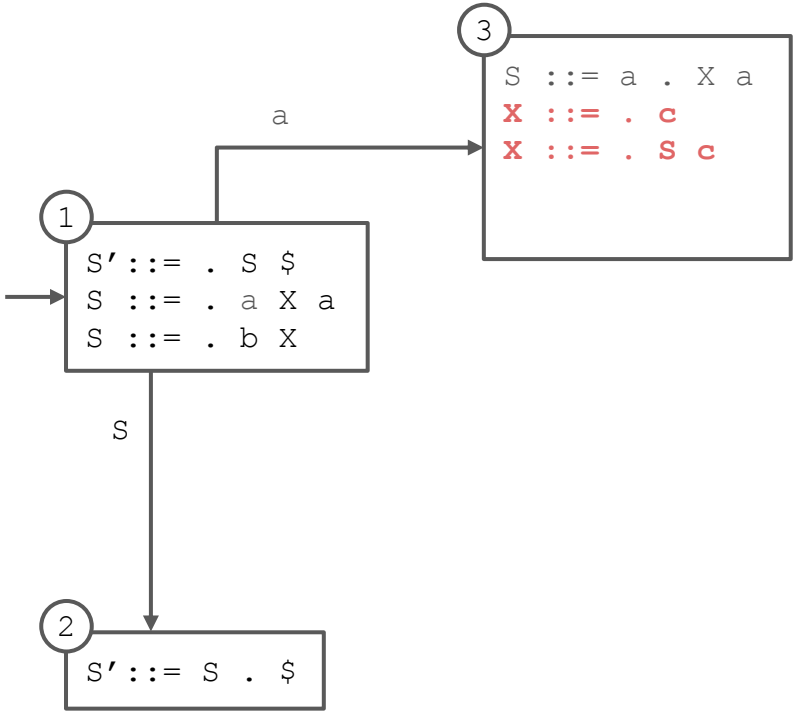
State Diagram Construction

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



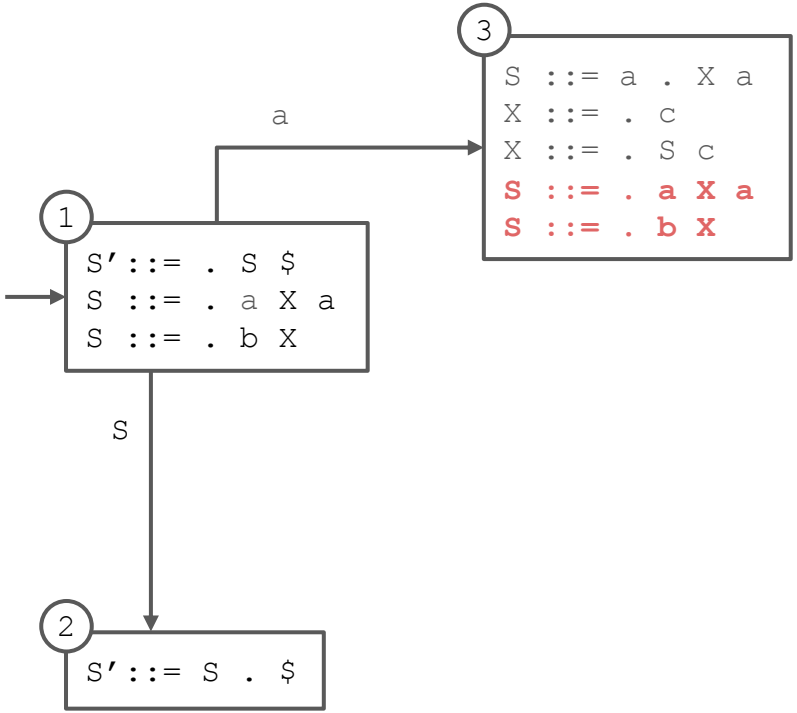
State Diagram Construction

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



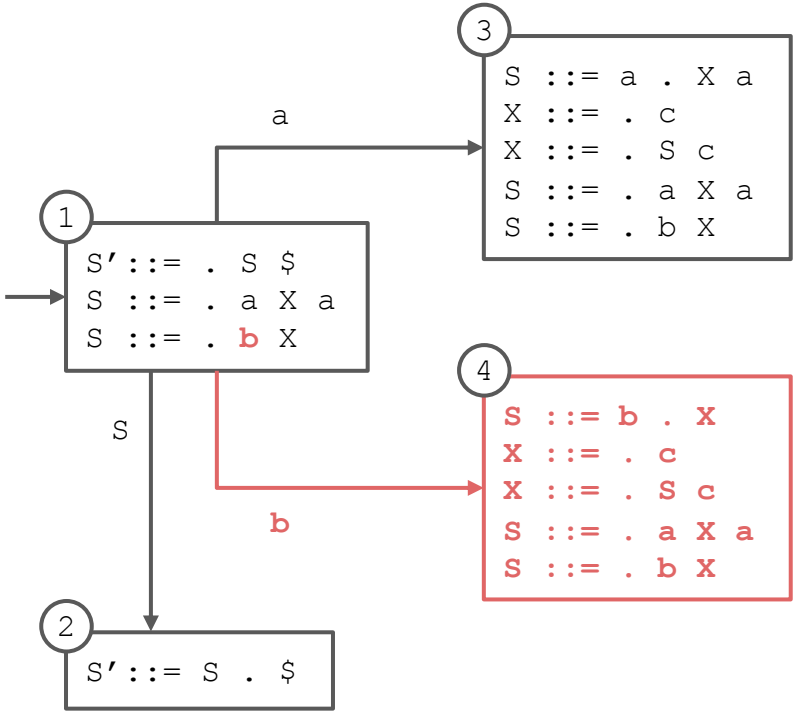
State Diagram Construction

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

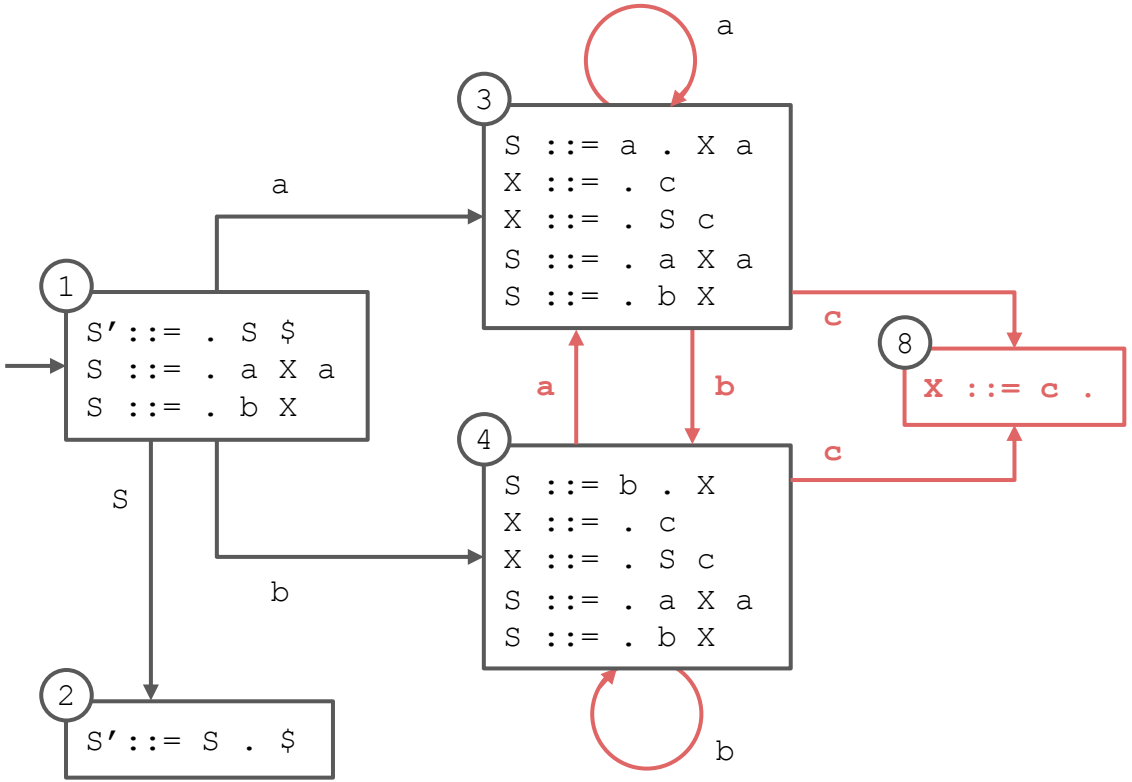


State Diagram Construction

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



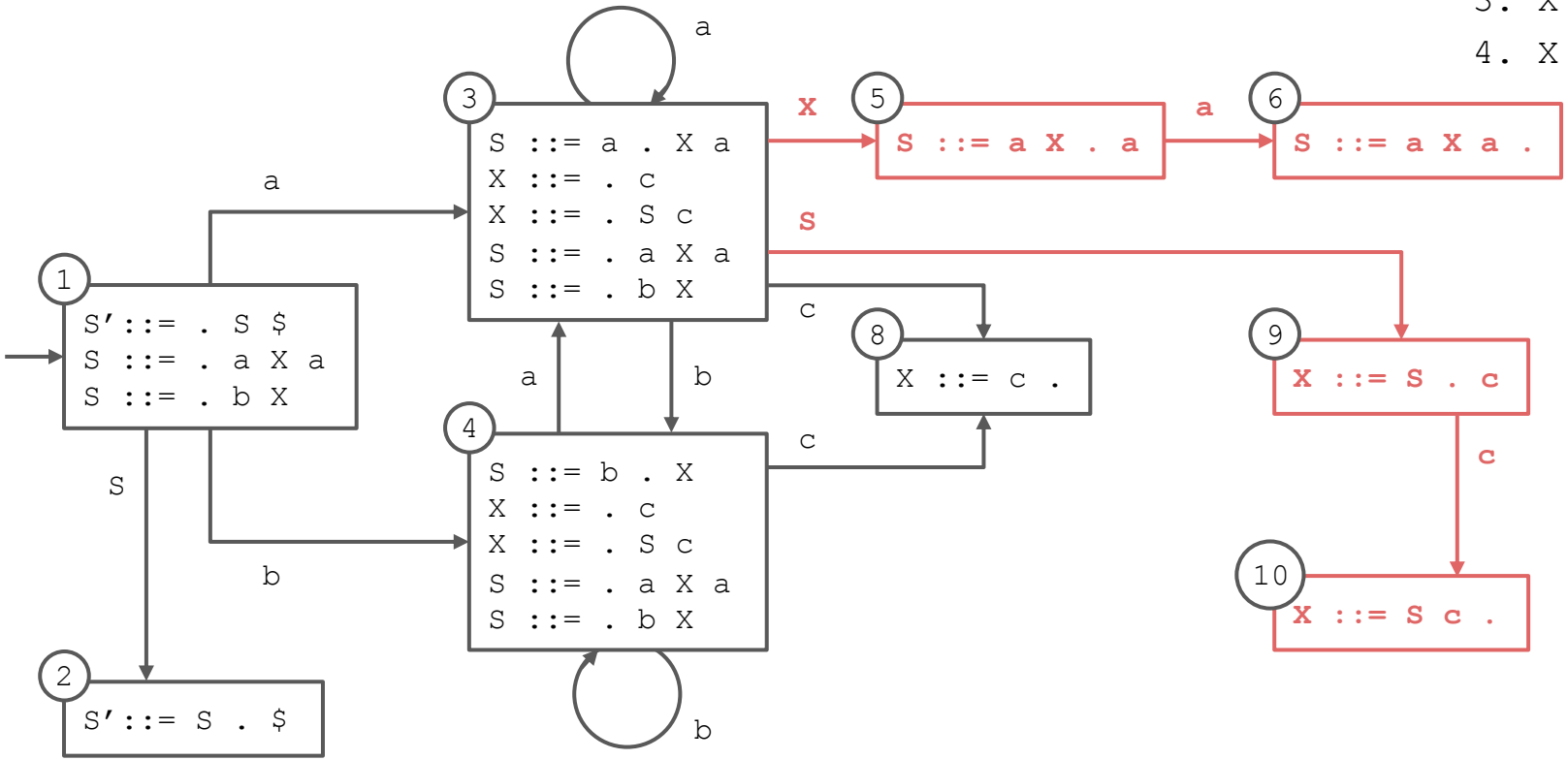
State Diagram Construction



0. S' ::= S $
1. S ::= a X a
2. S ::= b X
3. X ::= c
4. X ::= S c

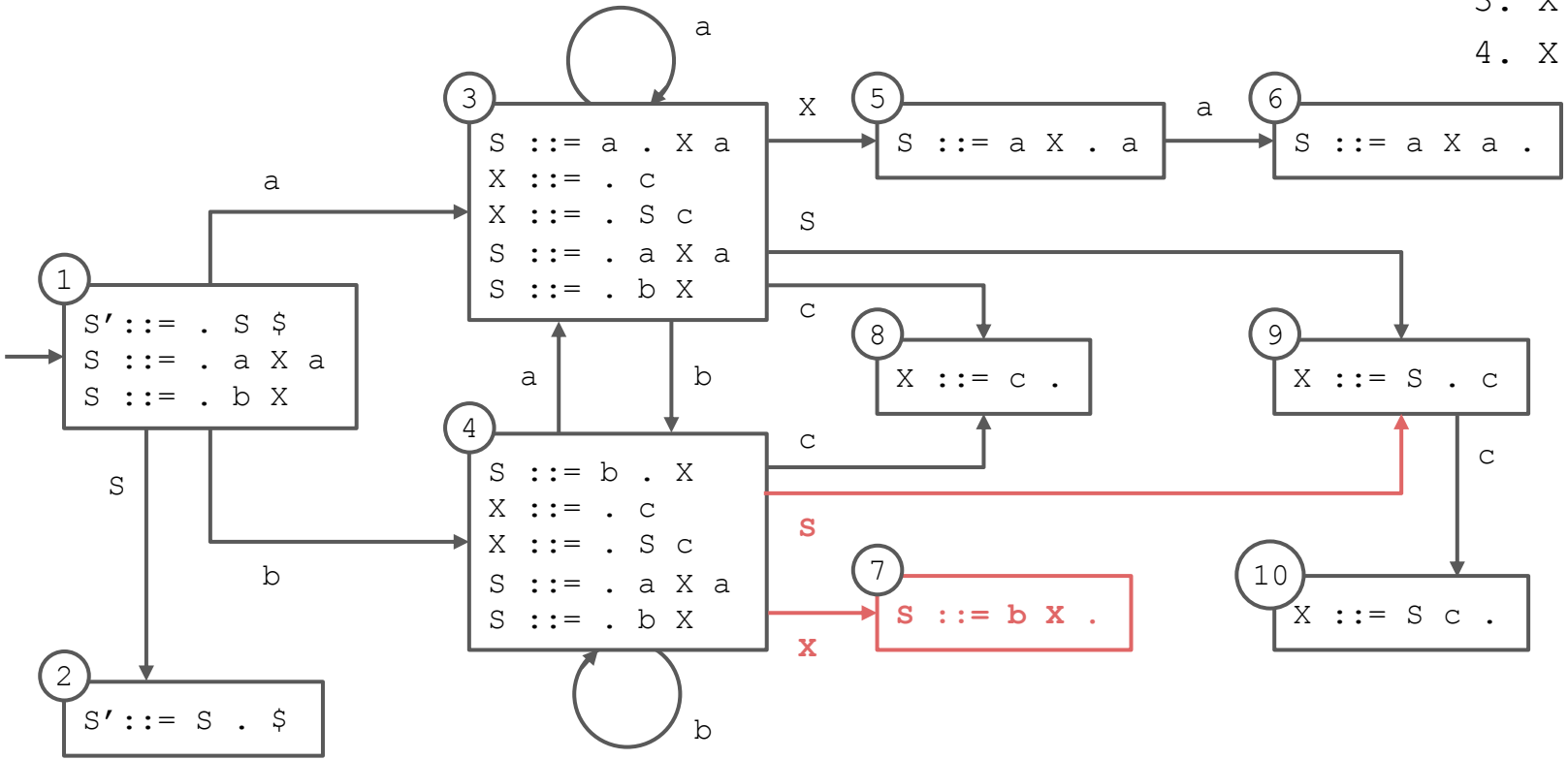
State Diagram Construction

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



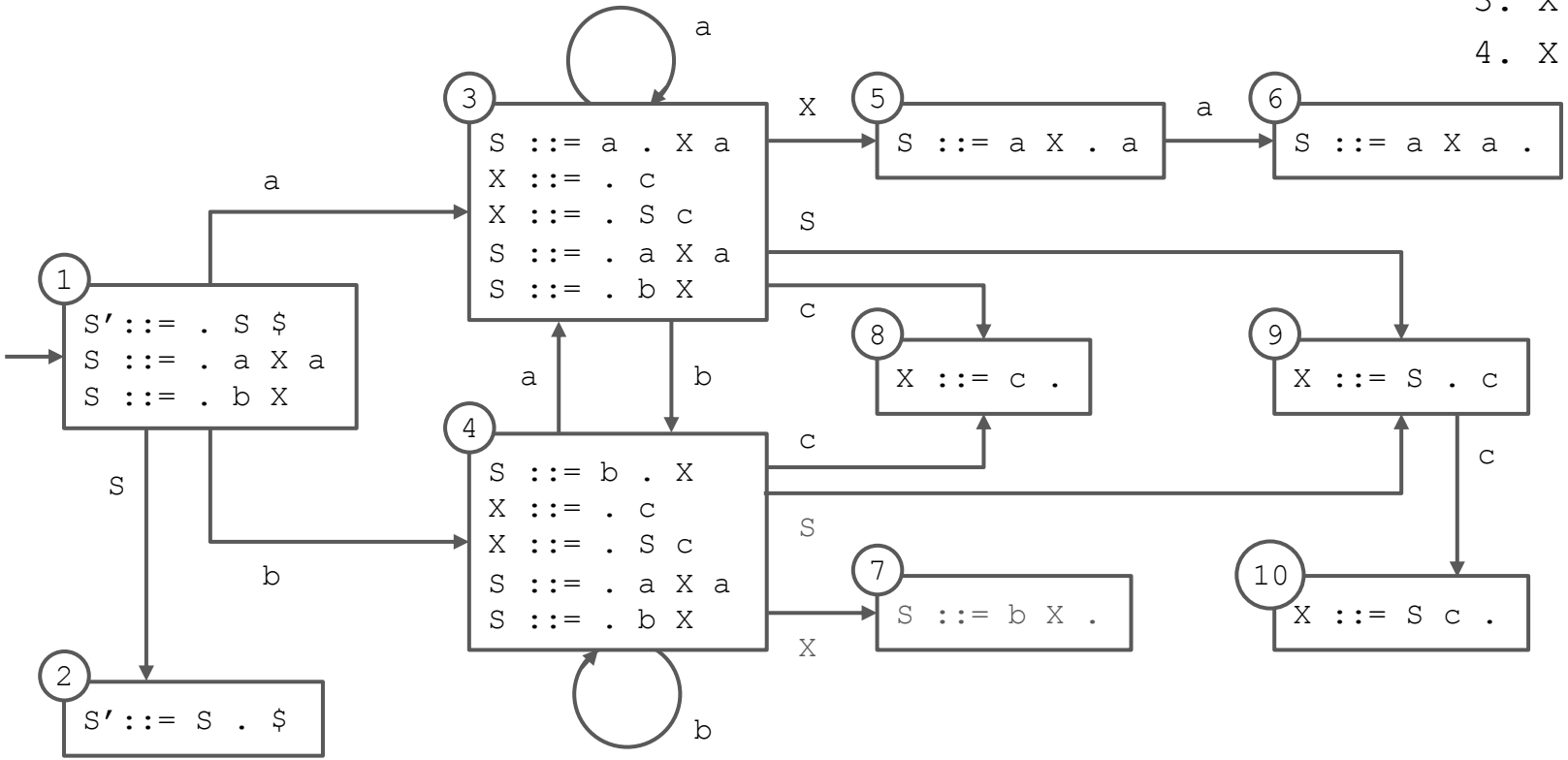
State Diagram Construction

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



State Diagram Construction

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



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"

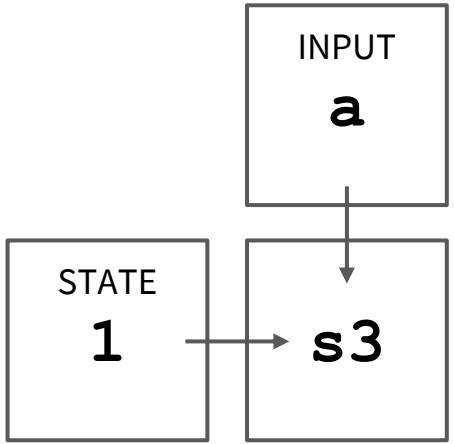
- occurs when the start symbol (S here) has been completed and there is no more input

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

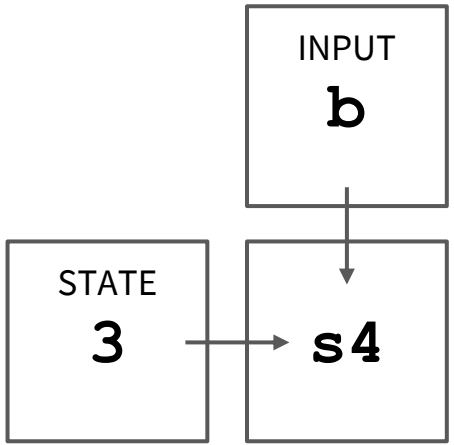
STACK	INPUT	ACTION
$ s_1	a b c c a $	

Parse Trace



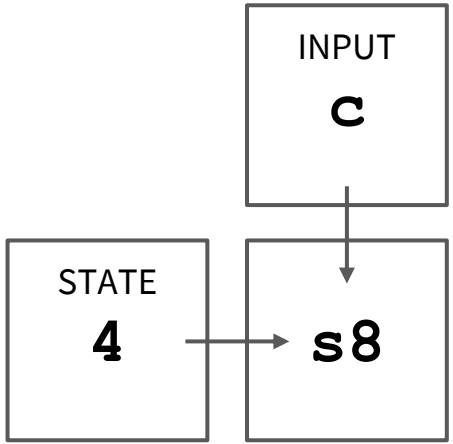
STACK	INPUT	ACTION
$ s_1	a b c c a $	SHIFT
$ s_1 a s_3	b c c a $	

Parse Trace



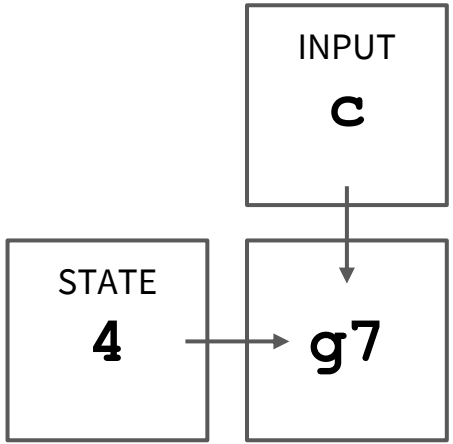
STACK	INPUT	ACTION
$ s_1	**a b c c a $**	SHIFT
$ s_1 a s_3	b c c a $	**SHIFT**
$ s_1 a s_3 b s_4	**c c a $**	

Parse Trace



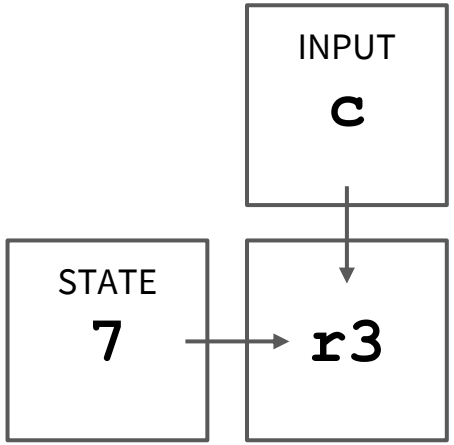
STACK	INPUT	ACTION
$ s_1	**a b c c a $**	SHIFT
$ s_1 a s_3	b c c a $	SHIFT
$ s_1 a s_3 b s_4	c c a $	**SHIFT**
$ s_1 a s_3 b s_4 c s_8	**c a $**	

Parse Trace



STACK	INPUT	ACTION
$ s_1	**a b c c a $**	SHIFT
$ s_1 a s_3	b c c a $	SHIFT
$ s_1 a s_3 b s_4	c c a $	SHIFT
$ s_1 a s_3 b s_4 c s_8	c a $	REDUCE
$ s_1 a s_3 b s_4 X s_7	**c a $**	

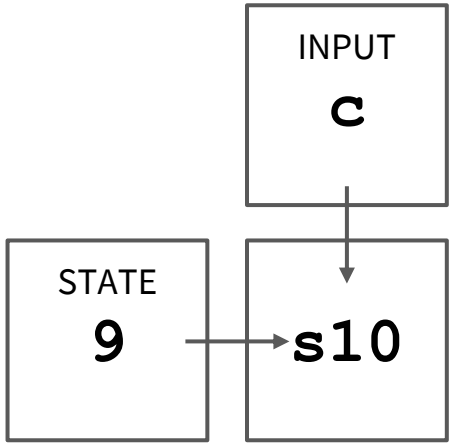
Parse Trace



3. S ::= b X

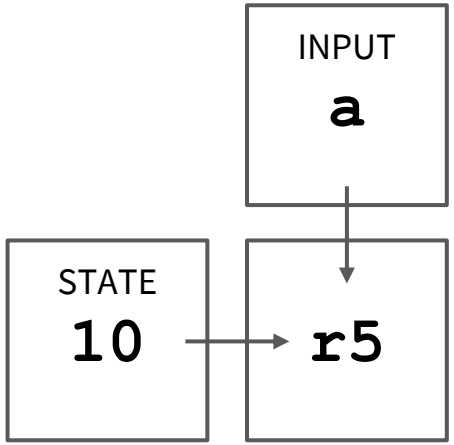
STACK	INPUT	ACTION
$ s_1	a b c c a $	SHIFT
$ s_1 a s_3	b c c a $	SHIFT
$ s_1 a s_3 b s_4	c c a $	SHIFT
$ s_1 a s_3 b s_4 c s_8	c a $	REDUCE
$ s_1 a s_3 b s_4 X s_7	c a $	REDUCE
$ s_1 a s_3 S s_9	c a $	

Parse Trace



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

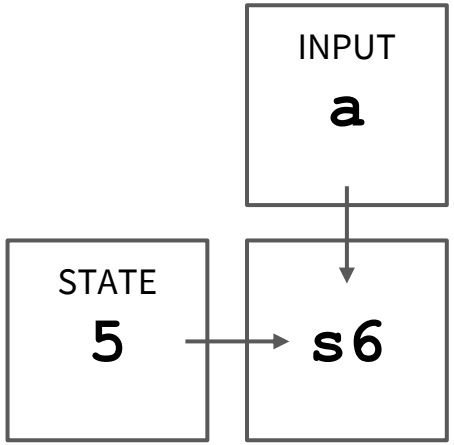
Parse Trace



5. X ::= S c

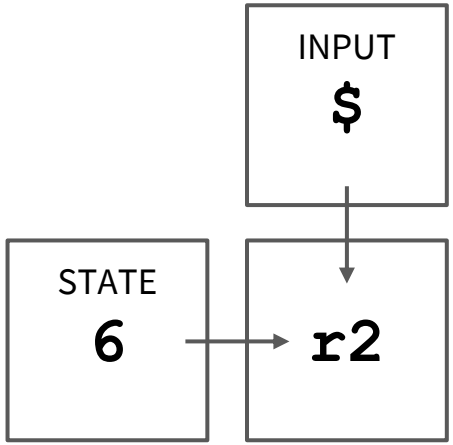
STACK	INPUT	ACTION
$ s_1	a b c c a $	SHIFT
$ s_1 a s_3	b c c a $	SHIFT
$ s_1 a s_3 b s_4	c c a $	SHIFT
$ s_1 a s_3 b s_4 c s_8	c a $	REDUCE
$ s_1 a s_3 b s_4 X s_7	c a $	REDUCE
$ s_1 a s_3 S s_9	c a $	SHIFT
$ s_1 a s_3 S s_9 c s_{10}	a $	REDUCE
$ s_1 a s_3 X s_5	a $	

Parse Trace



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

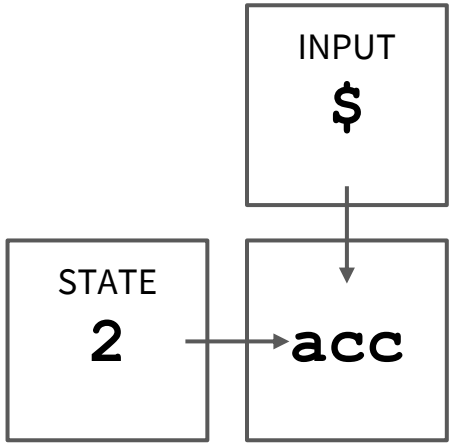
Parse Trace



2. S ::= a X a

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

Parse Trace



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