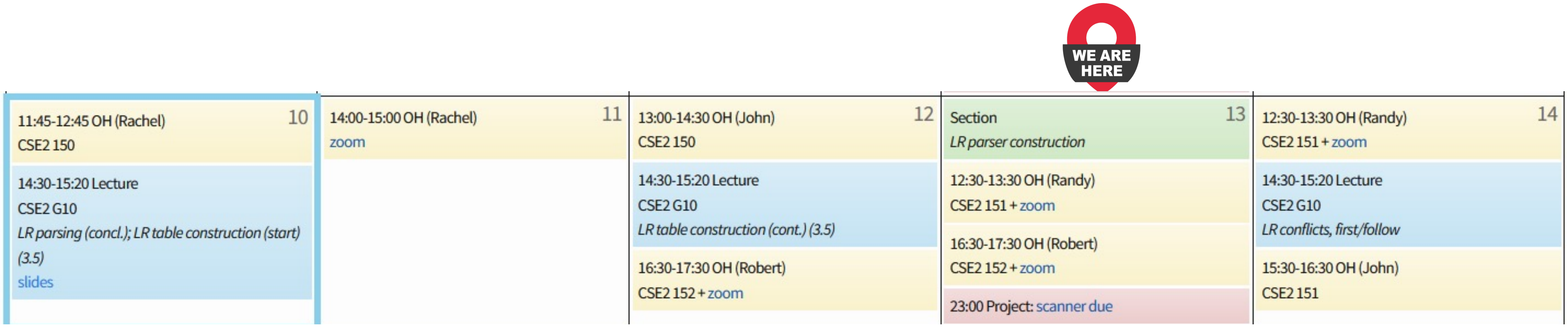
Section 3: LR Parsing

CSE 401/M501

Adapted from Spring 2021

Announcements

- Scanner is due tonight
    - Be sure to test, push, and tag!
- Every person has 4 project late days and 4 assignment late days
    - Up to 2 can be used per assignment
    - Each late day gives an extra 24 hour chunk (including weekend days)
    - We recommend not using your late days this early if possible!
    - **Submitting project components a day late uses a project late day from each partner!**



10	11	12	13	14
11:45-12:45 OH (Rachel) CSE2 150	14:00-15:00 OH (Rachel) zoom	13:00-14:30 OH (John) CSE2 150	Section *LR parser construction*	12:30-13:30 OH (Randy) CSE2 151 + zoom
14:30-15:20 Lecture CSE2 G10 *LR parsing (concl.); LR table construction (start) (3.5)* slides		14:30-15:20 Lecture CSE2 G10 *LR table construction (cont.) (3.5)*	12:30-13:30 OH (Randy) CSE2 151 + zoom	14:30-15:20 Lecture CSE2 G10 *LR conflicts, first/follow*
		16:30-17:30 OH (Robert) CSE2 152 + zoom	16:30-17:30 OH (Robert) CSE2 152 + zoom	15:30-16:30 OH (John) CSE2 151
			23:00 Project: scanner due	

Agenda

- (Fast) LR terminology review
- Worksheet

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 … **=> a$\underline{\texttt{A}}$bcde => a$\underline{\texttt{b}}$bcde**, using the production **A ::= b:**
        - The production ‘**A ::= $\underline{\texttt{b}}$**’ at index 1 would be a handle of **a$\underline{\texttt{b}}$bcde**

Get Your LR Jargon On - Example

Shift-Reduce Example

S ::= a*AB*e
A ::= *A*bc | b
B ::= d

Frontier

Stack	Input	Action
$	abbcde$	*shift*
$a	bbcde$	*shift*
$ab	bcde$	*reduce*
$aA	bcde$	*shift*
$aAb	cde$	*shift*
$aAbc	de$	*reduce*
$aA	de$	*shift*
$aAd	e$	*reduce*
$aAB	e$	*shift*
$aABe	$	*reduce*
$S	$	*accept*

Get Your LR Jargon On - Example

Shift-Reduce Example

S ::= a*AB*e
A ::= *A*bc | b
B ::= d

Stack	Input	Action
$	abbcde$	*shift*
$a	bbcde$	*shift*
$ab	bcde$	*reduce*
$aA	bcde$	*shift*
$aAb	cde$	*shift*
$aAbc	de$	*reduce*
$aA	de$	*shift*
$aAd	e$	*reduce*
$aAB	e$	*shift*
$aABe	$	*reduce*
$S	$	*accept*

Sentential Forms

Get Your LR Jargon On - Example

Shift-Reduce Example

S ::= a*AB*e
A ::= *A*bc | b
B ::= d

Handles	Stack	Input	Action
	$	abbcde$	*shift*
	$a	bbcde$	*shift*
A ::= b at index 2	$ab	bcde$	*reduce*
	$aA	bcde$	*shift*
	$aAb	cde$	*shift*
A ::= Abc at index 4	$aAbc	de$	*reduce*
	$aA	de$	*shift*
B ::= d at index 3	$aAd	e$	*reduce*
	$aAB	e$	*shift*
S ::= aABe at index 4	$aABe	$	*reduce*
	$S	$	*accept*

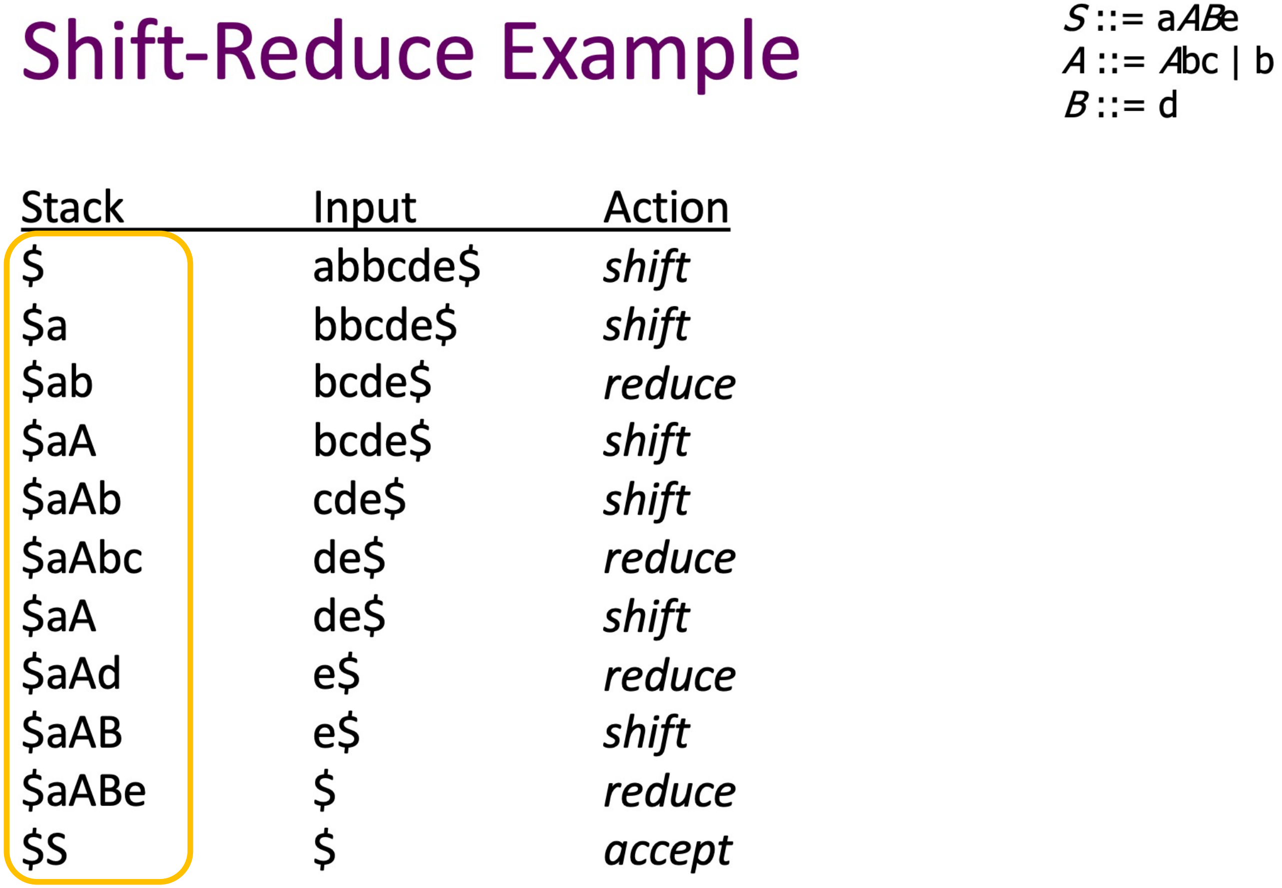
A Little Bit More Jargon

- Viable Prefix
  - The prefixes of a right sentential form that do not extend beyond the end of its handle
  - Perhaps less confusing -> the set of prefixes of strings that can appear on the stack of a shift-reduce parser

A Little Bit More Jargon

- Viable Prefix
  - The prefixes of a right sentential form that do not extend beyond the end of its handle
  - Perhaps less confusing -> the set of prefixes of strings that can appear on the stack of a shift-reduce parser
- Item
  - A marked production (a production with a '.' in it)
    - [A ::= .XY], [A ::= X.Y], [A ::= XY.]

Get Your LR Jargon On - Example

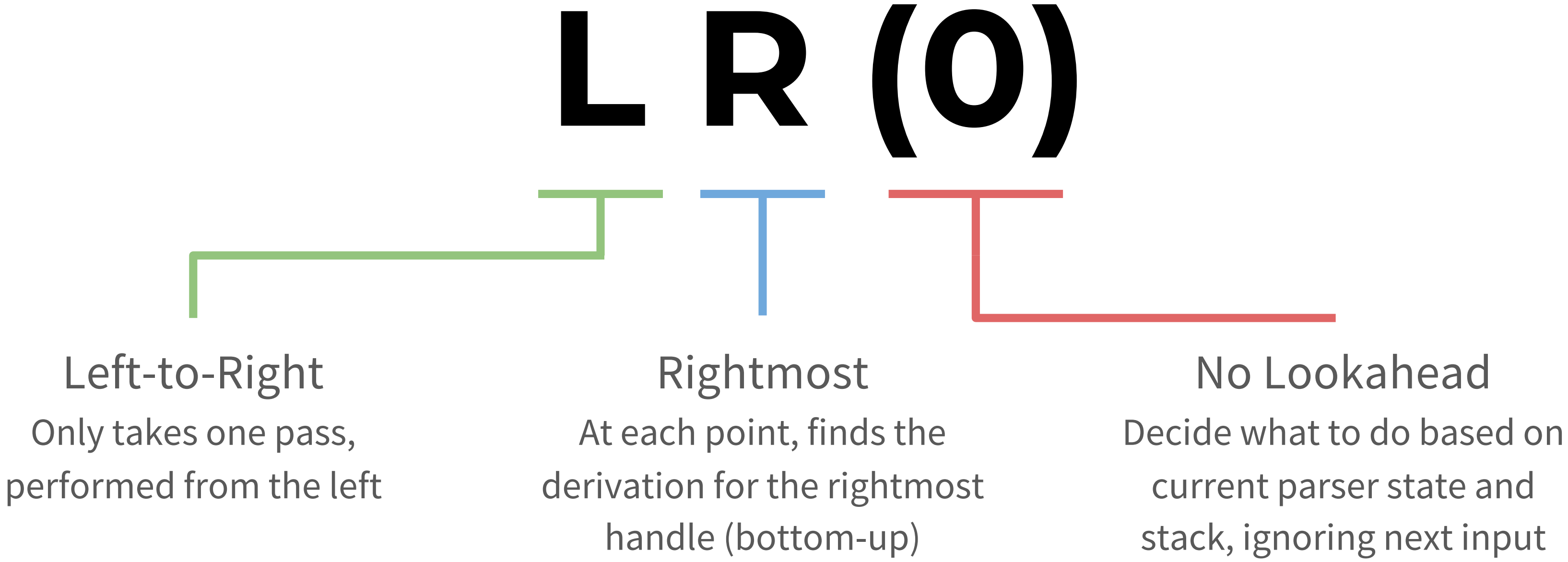


Shift-Reduce Example

S ::= a*AB*e
A ::= *A*bc | b
B ::= d

Stack	Input	Action
$	abbcde$	*shift*
$a	bbcde$	*shift*
$ab	bcde$	*reduce*
$aA	bcde$	*shift*
$aAb	cde$	*shift*
$aAbc	de$	*reduce*
$aA	de$	*shift*
$aAd	e$	*reduce*
$aAB	e$	*shift*
$aABe	$	*reduce*
$S	$	*accept*

Viable Prefix
(all prefixes of these strings are also viable prefixes)



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

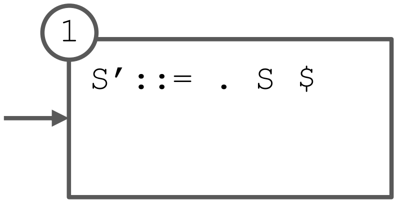
Decide what to do based on current parser state and stack, ignoring next input

Problem 1 (On Worksheet)

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

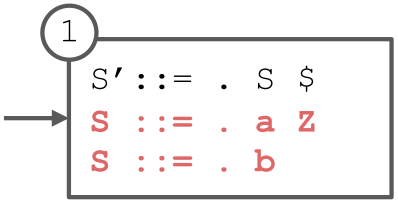
State Diagram Construction

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



State Diagram Construction

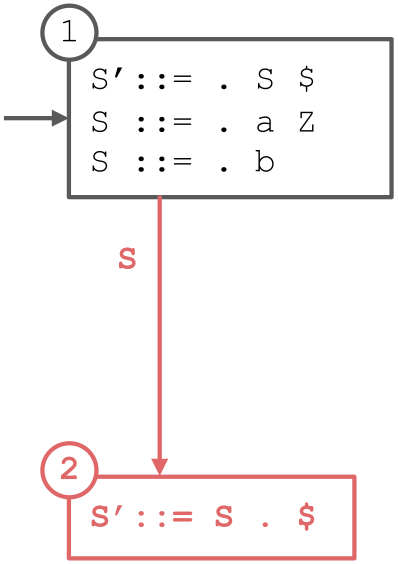
```
0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S
```



```
S’::= . S $
S ::= . a Z
S ::= . b
```

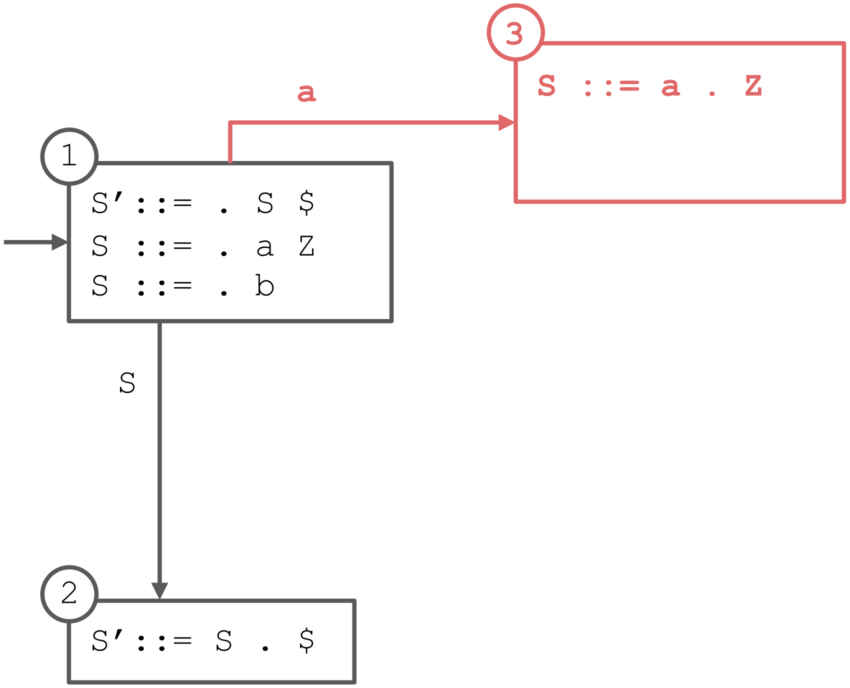
State Diagram Construction

```
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
```



1

```
S’::= . S $
S ::= . a Z
S ::= . b
```

a

3

```
S ::= a . Z
```

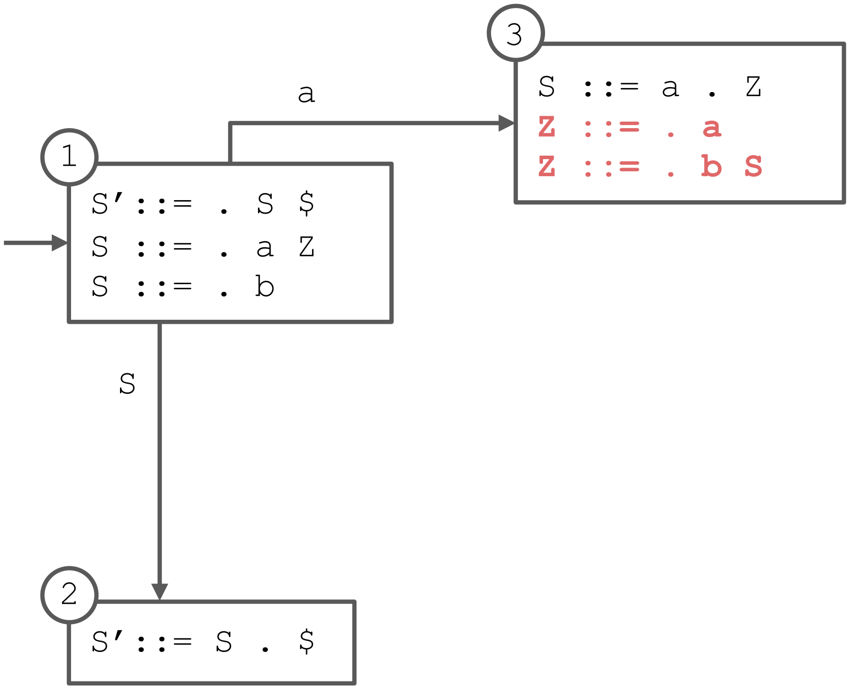
S

2

```
S’::= S . $
```

State Diagram Construction

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



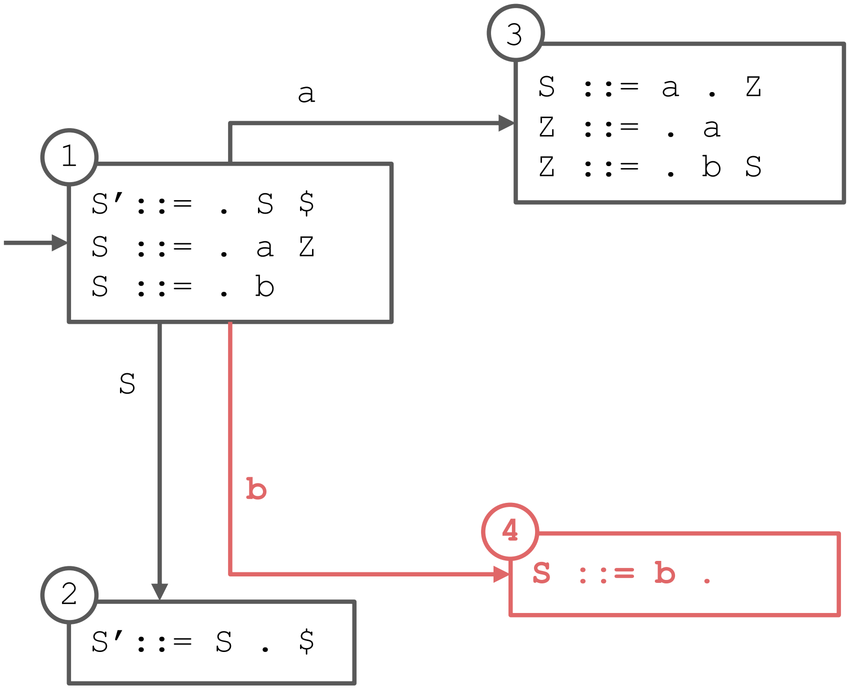
S’::= . S $
S ::= . a Z
S ::= . b

S ::= a . Z
Z ::= . a
Z ::= . b S

S’::= S . $

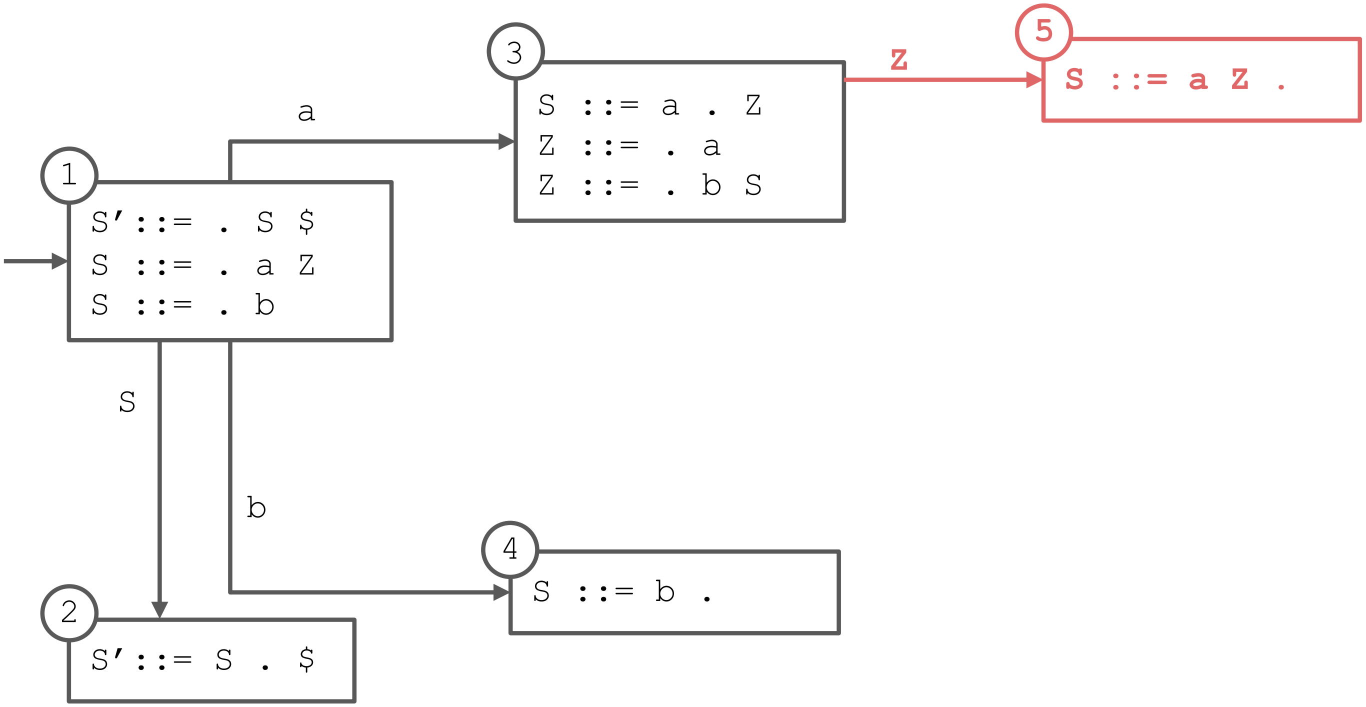
State Diagram Construction

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



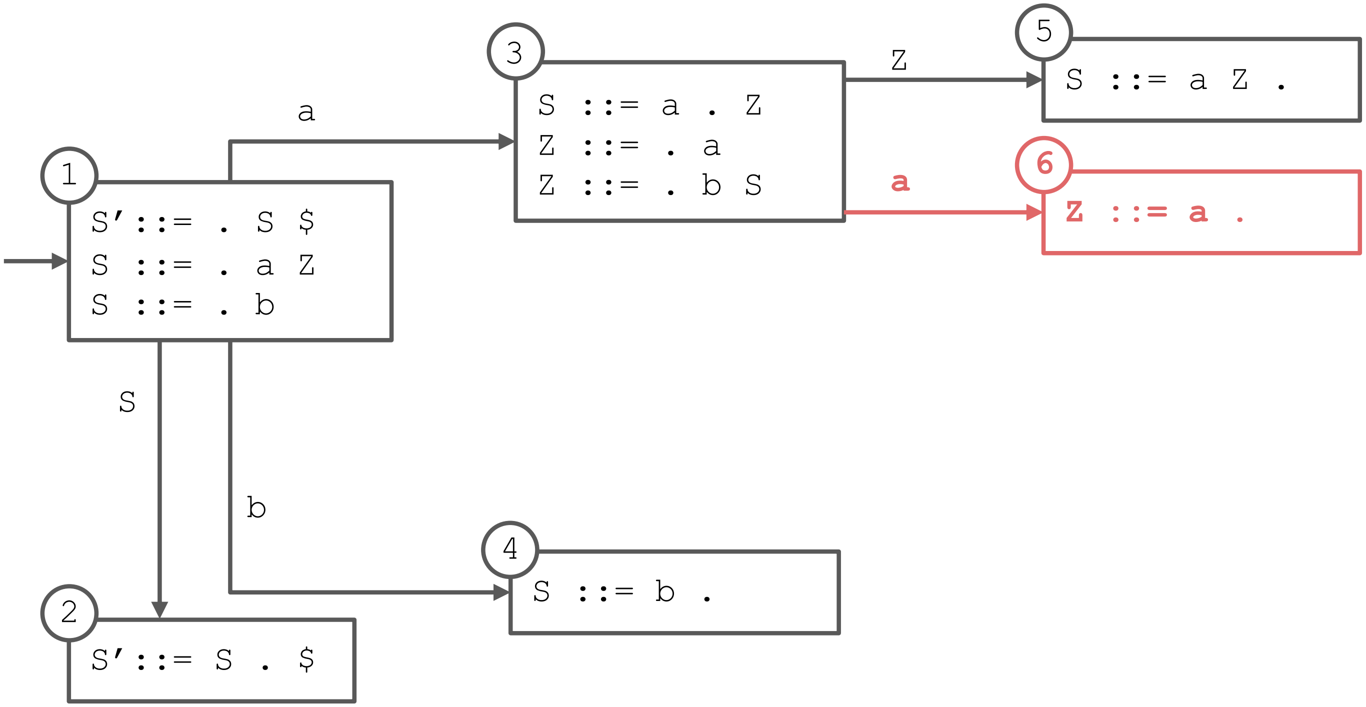
State Diagram Construction

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



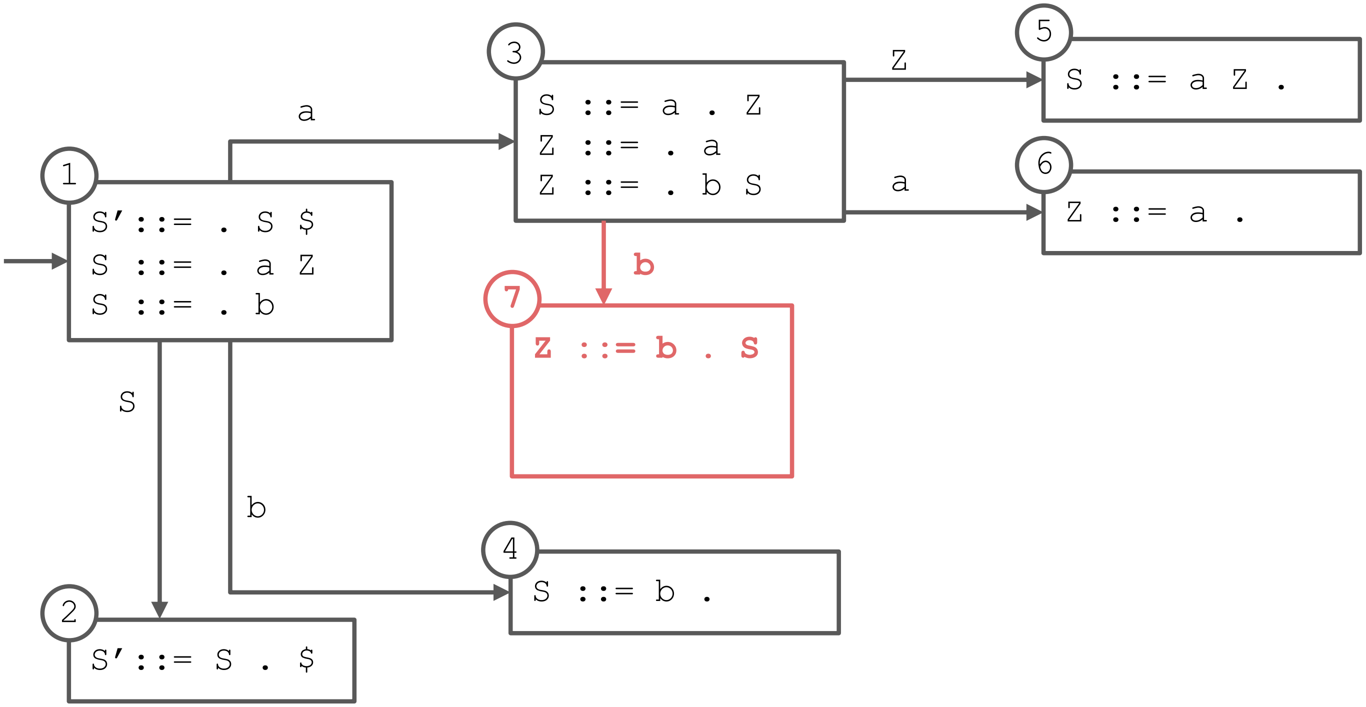
State Diagram Construction

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



1
S′::= . S $
S ::= . a Z
S ::= . b

2
S′::= S . $

3
S ::= a . Z
Z ::= . a
Z ::= . b S

4
S ::= b .

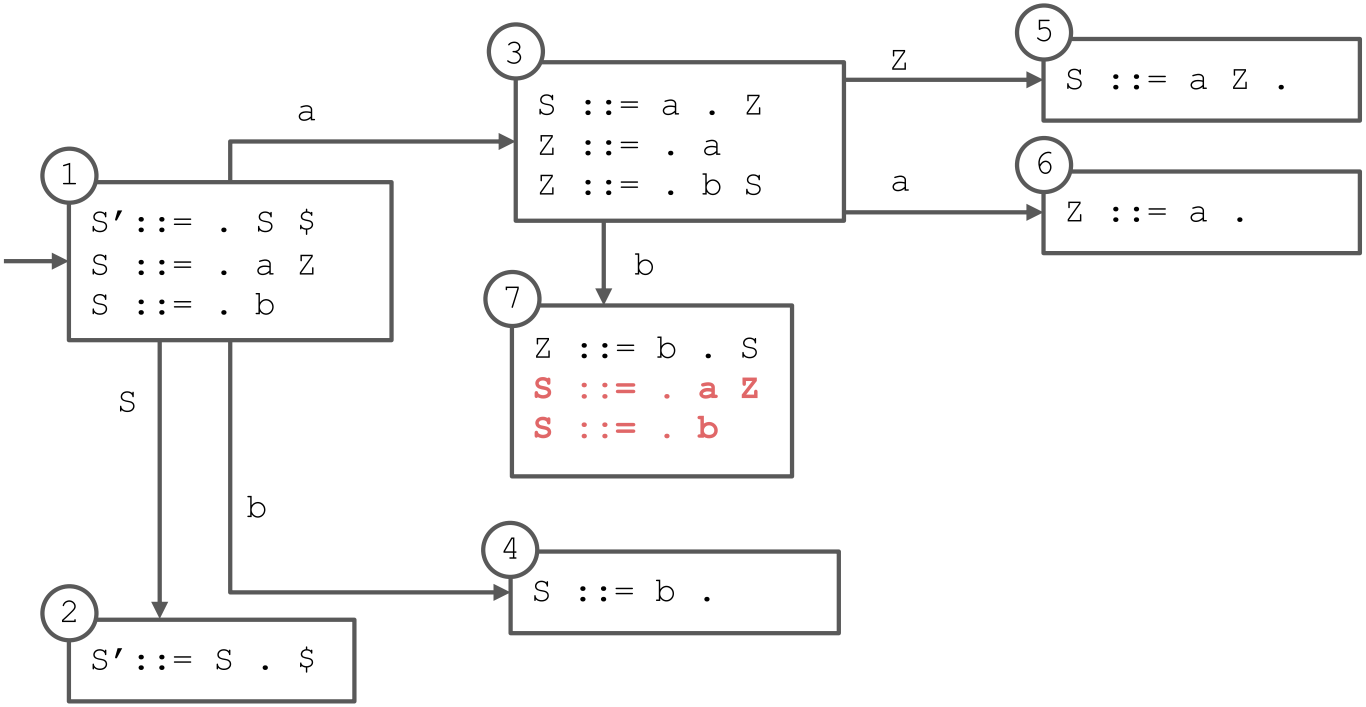
5
S ::= a Z .

6
Z ::= a .

7
Z ::= b . S

State Diagram Construction

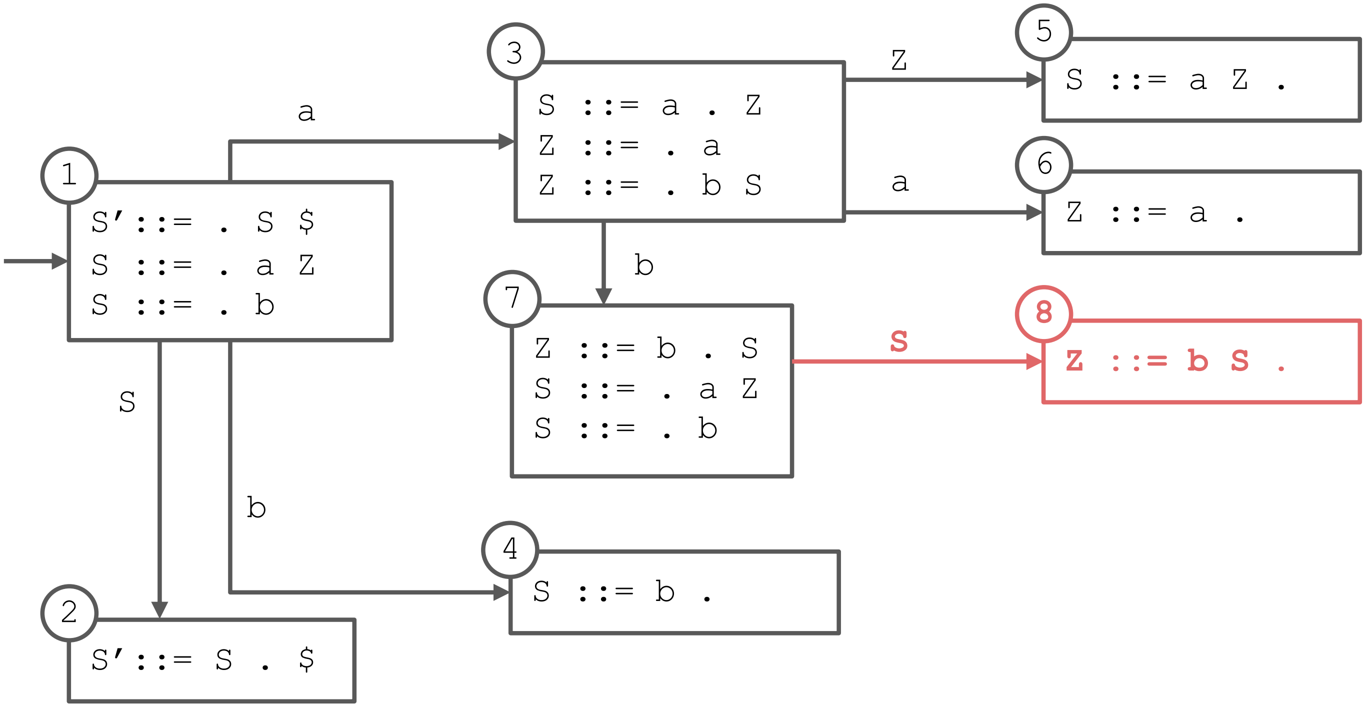
```
0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S
```



```
1
S’::= . S $
S ::= . a Z
S ::= . b

2
S’::= S . $

3
S ::= a . Z
Z ::= . a
Z ::= . b S

4
S ::= b .

5
S ::= a Z .

6
Z ::= a .

7
Z ::= b . S
S ::= . a Z
S ::= . b
```

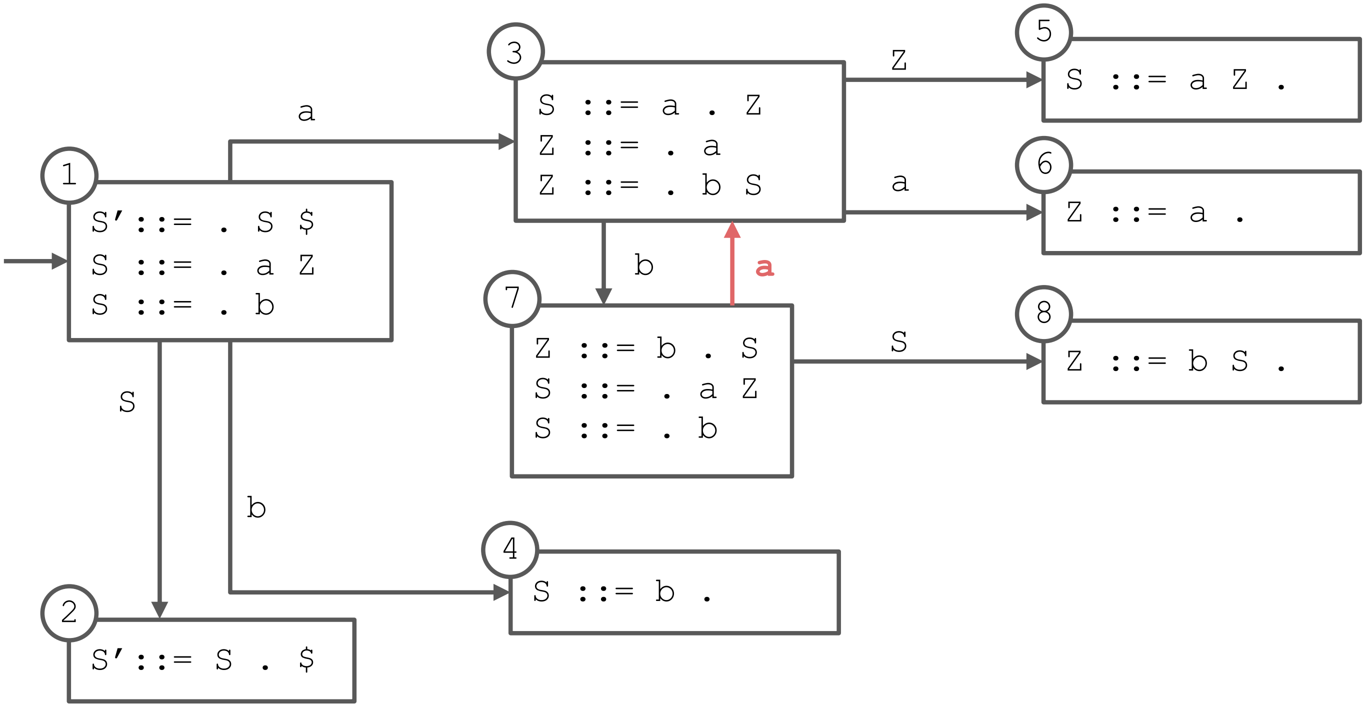
State Diagram Construction

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
```



1
S’::= . S $
S ::= . a Z
S ::= . b

2
S’::= S . $

3
S ::= a . Z
Z ::= . a
Z ::= . b S

4
S ::= b .

5
S ::= a Z .

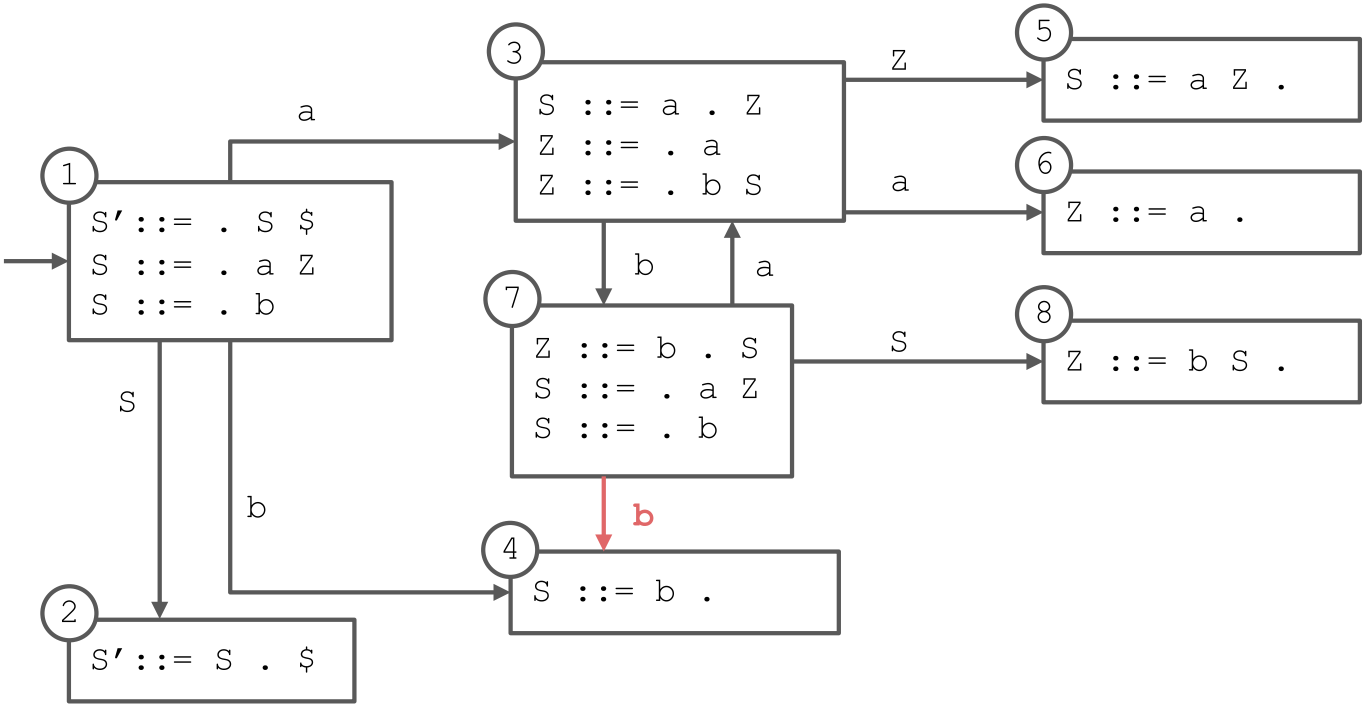
6
Z ::= a .

7
Z ::= b . S
S ::= . a Z
S ::= . b

8
Z ::= b S .

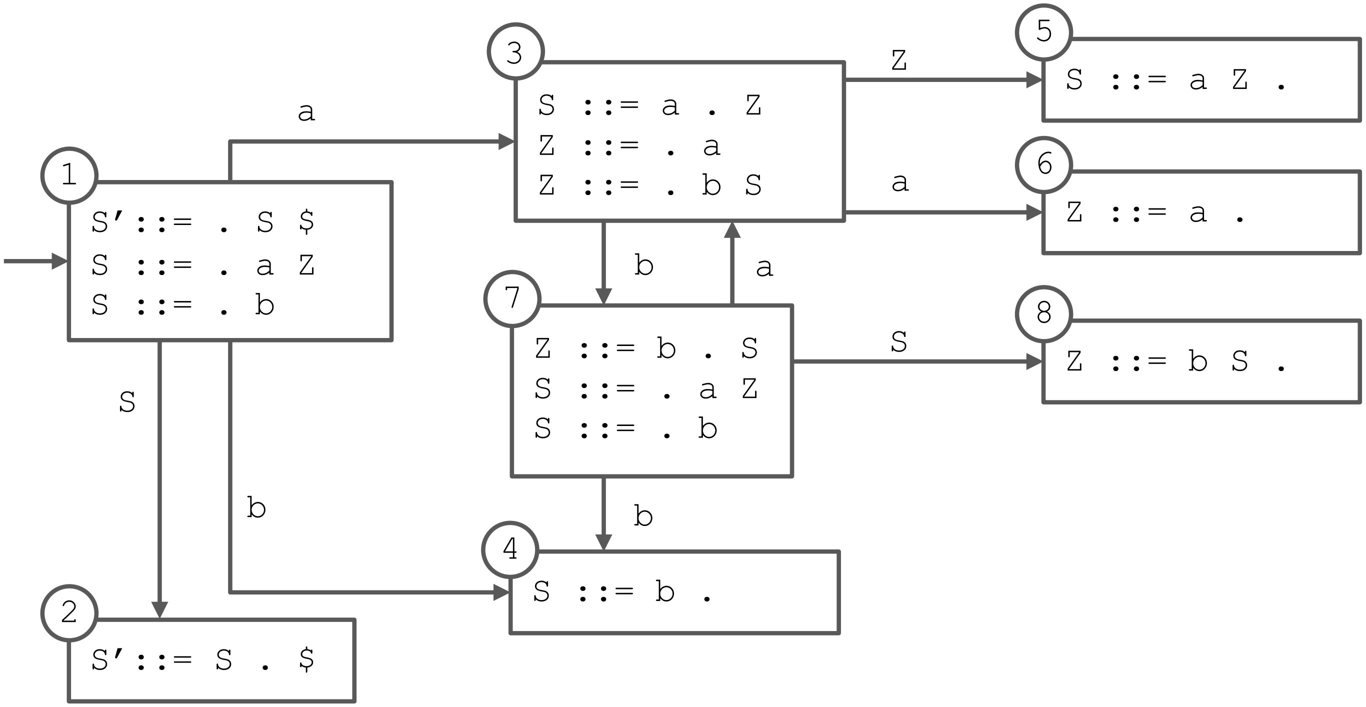
State Diagram Construction

```
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
```



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 #”

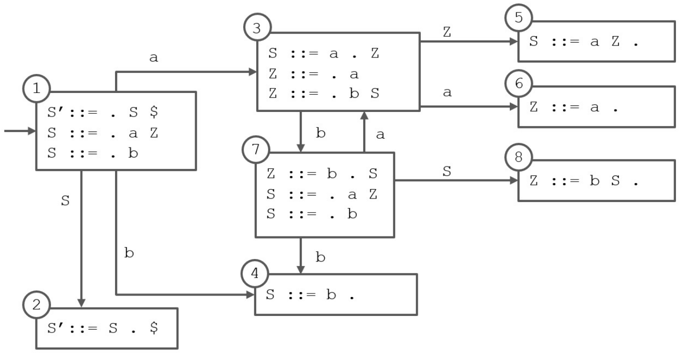
- occurs when there is a transition on a nonterminal

`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		

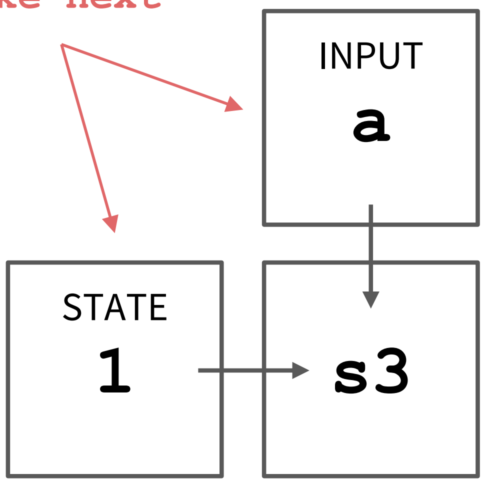
Parse Trace



STACK	INPUT	ACTION
$ 1	a b a b b $	

Parse Trace

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



INPUT

a

STATE

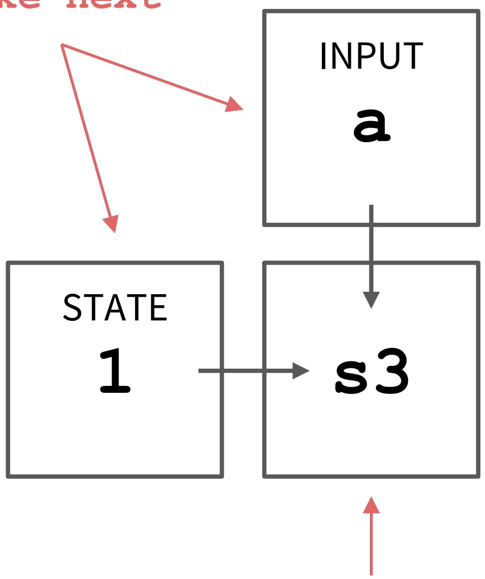
1

s3

STACK	INPUT	ACTION
$ 1	a b a b b $	SHIFT
$ 1 a	b a b b $	

Parse Trace

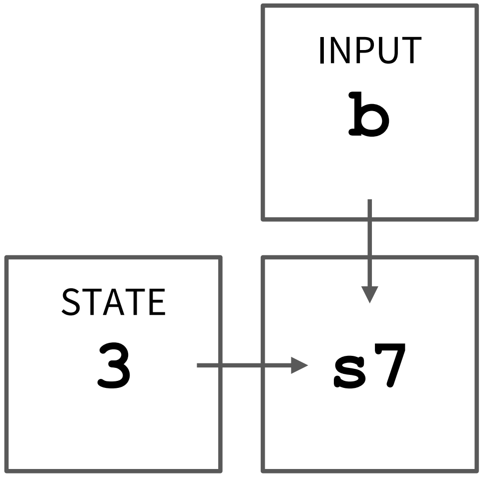
Row and column of table to look up: decides what action to take next



Shift, and enter state 3

STACK	INPUT	ACTION
$ 1	a b a b b $	SHIFT
$ 1 a 3	b a b b $	

Parse Trace



INPUT

b

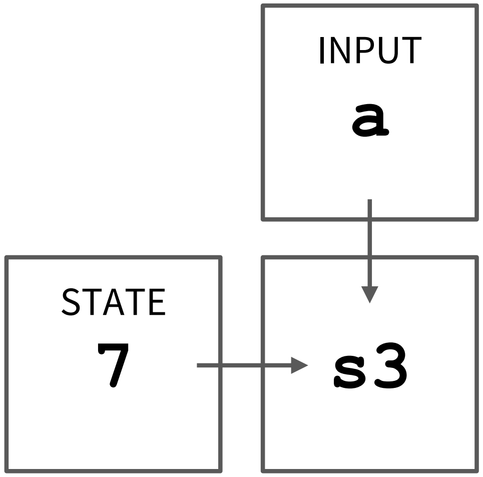
STATE

3

s7

STACK	INPUT	ACTION
$ 1	a b a b b $	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	

Parse Trace



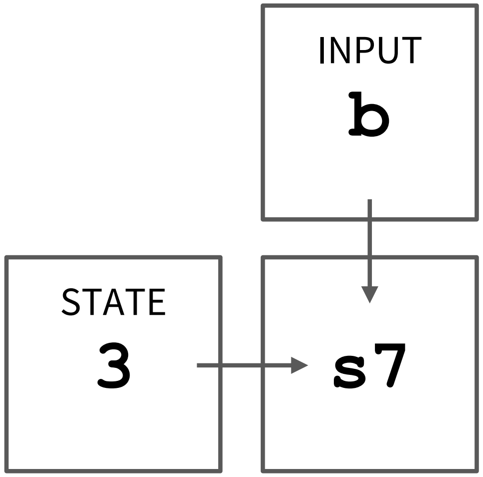
INPUT
a

STATE
7

s3

STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	**SHIFT**
$ 1 a 3 b 7 a 3	**b b $**	

Parse Trace



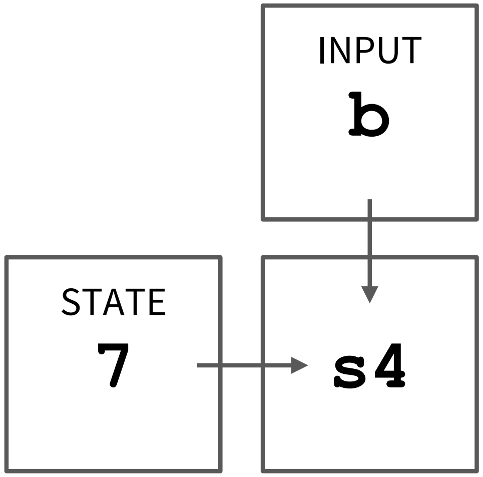
INPUT
b

STATE
3

s7

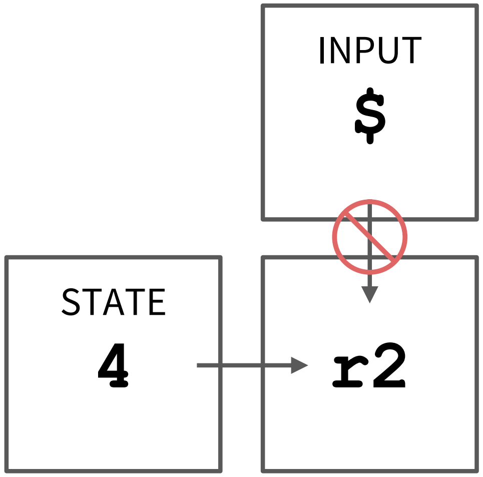
STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	**SHIFT**
$ 1 a 3 b 7 a 3 b 7	**b $**	

Parse Trace



STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	SHIFT
$ 1 a 3 b 7 a 3 b 7	b $	**SHIFT**
$ 1 a 3 b 7 a 3 b 7 b 4	**$**	

Parse Trace



INPUT

$

STATE

4

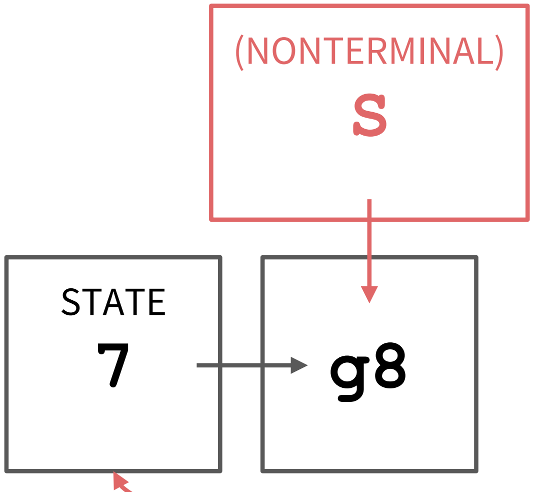
r2

`2. S ::= b`

STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	SHIFT
$ 1 a 3 b 7 a 3 b 7	b $	SHIFT
$ 1 a 3 b 7 a 3 b 7 b 4	$	**REDUCE**
$ 1 a 3 b 7 a 3 b 7 S	**$**	

For LR(0), the input doesn't technically matter here

Parse Trace

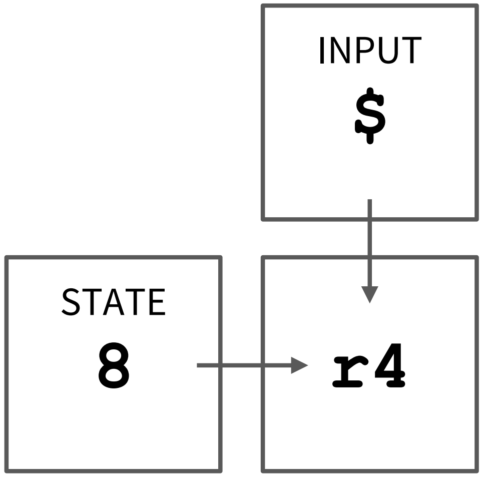


STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	SHIFT
$ 1 a 3 b 7 a 3 b 7	b $	SHIFT
$ 1 a 3 b 7 a 3 b 7 b 4	$	REDUCE
$ 1 a 3 b 7 a 3 b 7 S 8	$	

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

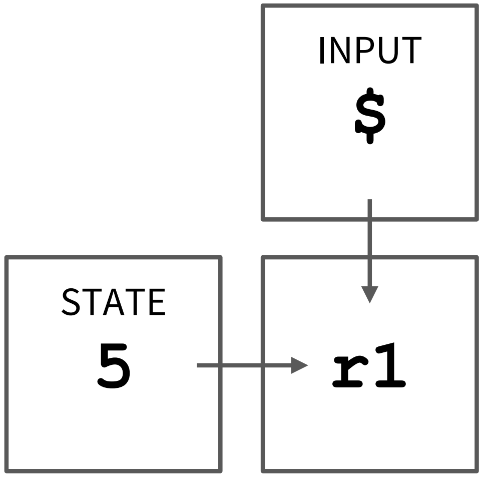


4. Z ::= b S

(and GOTO step:
s3 & Z -> g5)

STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	SHIFT
$ 1 a 3 b 7 a 3 b 7	b $	SHIFT
$ 1 a 3 b 7 a 3 b 7 b 4	$	REDUCE
$ 1 a 3 b 7 a 3 b 7 S 8	$	**REDUCE**
$ 1 a 3 b 7 a 3 Z 5	**$**	

Parse Trace

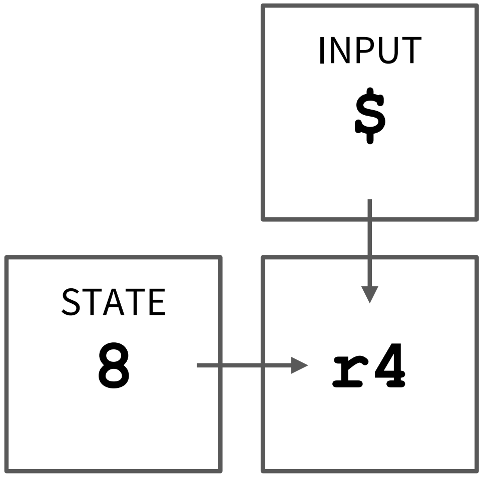


1. S ::= a Z

(and GOTO step:
s7 & S -> g8)

STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	SHIFT
$ 1 a 3 b 7 a 3 b 7	b $	SHIFT
$ 1 a 3 b 7 a 3 b 7 b 4	$	REDUCE
$ 1 a 3 b 7 a 3 b 7 S 8	$	REDUCE
$ 1 a 3 b 7 a 3 Z 5	$	**REDUCE**
$ 1 a 3 b 7 S 8	**$**	

Parse Trace



INPUT

$

STATE

8

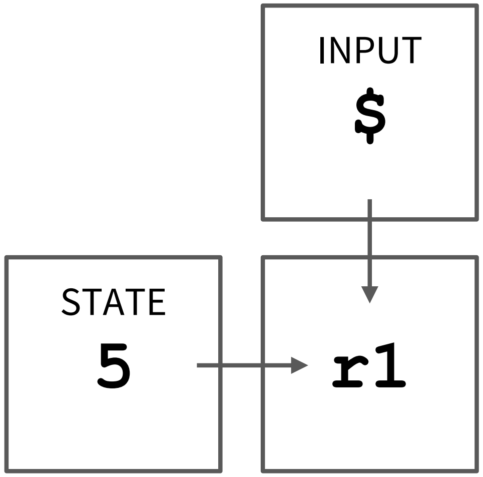
r4

4. Z ::= b S

(and GOTO step:
s3 & Z -> g5)

STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	SHIFT
$ 1 a 3 b 7 a 3 b 7	b $	SHIFT
$ 1 a 3 b 7 a 3 b 7 b 4	$	REDUCE
$ 1 a 3 b 7 a 3 b 7 S 8	$	REDUCE
$ 1 a 3 b 7 a 3 Z 5	$	REDUCE
$ 1 a 3 b 7 S 8	$	**REDUCE**
$ 1 a 3 Z 5	**$**	

Parse Trace

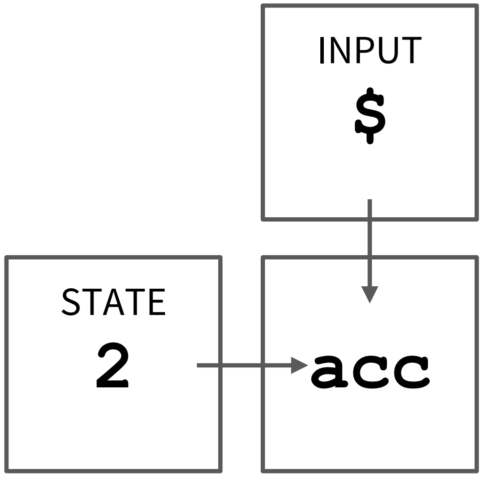


`1. S ::= a Z`

`(and GOTO step:`
`s1 & S -> g2)`

STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	SHIFT
$ 1 a 3 b 7 a 3 b 7	b $	SHIFT
$ 1 a 3 b 7 a 3 b 7 b 4	$	REDUCE
$ 1 a 3 b 7 a 3 b 7 S 8	$	REDUCE
$ 1 a 3 b 7 a 3 Z 5	$	REDUCE
$ 1 a 3 b 7 S 8	$	REDUCE
$ 1 a 3 Z 5	$	**REDUCE**
$ 1 S 2	**$**	

Parse Trace



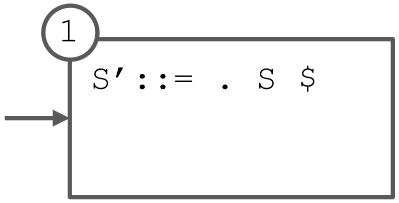
STACK	INPUT	ACTION
$ 1	**a b a b b $**	SHIFT
$ 1 a 3	b a b b $	SHIFT
$ 1 a 3 b 7	a b b $	SHIFT
$ 1 a 3 b 7 a 3	b b $	SHIFT
$ 1 a 3 b 7 a 3 b 7	b $	SHIFT
$ 1 a 3 b 7 a 3 b 7 b 4	$	REDUCE
$ 1 a 3 b 7 a 3 b 7 S 8	$	REDUCE
$ 1 a 3 b 7 a 3 Z 5	$	REDUCE
$ 1 a 3 b 7 S 8	$	REDUCE
$ 1 a 3 Z 5	$	REDUCE
$ 1 S 2	$	ACCEPT

Problem 2 (On Worksheet)

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

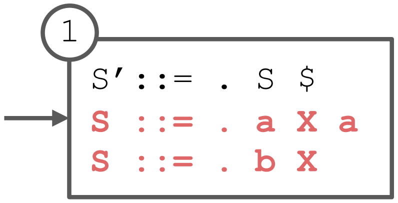
State Diagram Construction

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



State Diagram Construction

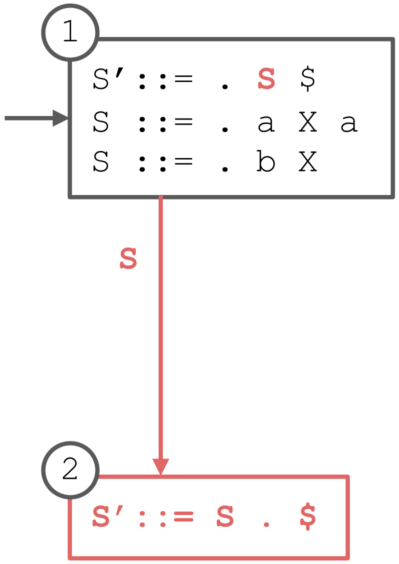
```
0. S’::= S $
1. S ::= a X a
2. S ::= b X
3. X ::= c
4. X ::= S c
```



```
S’::= . S $
S ::= . a X a
S ::= . b X
```

State Diagram Construction

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

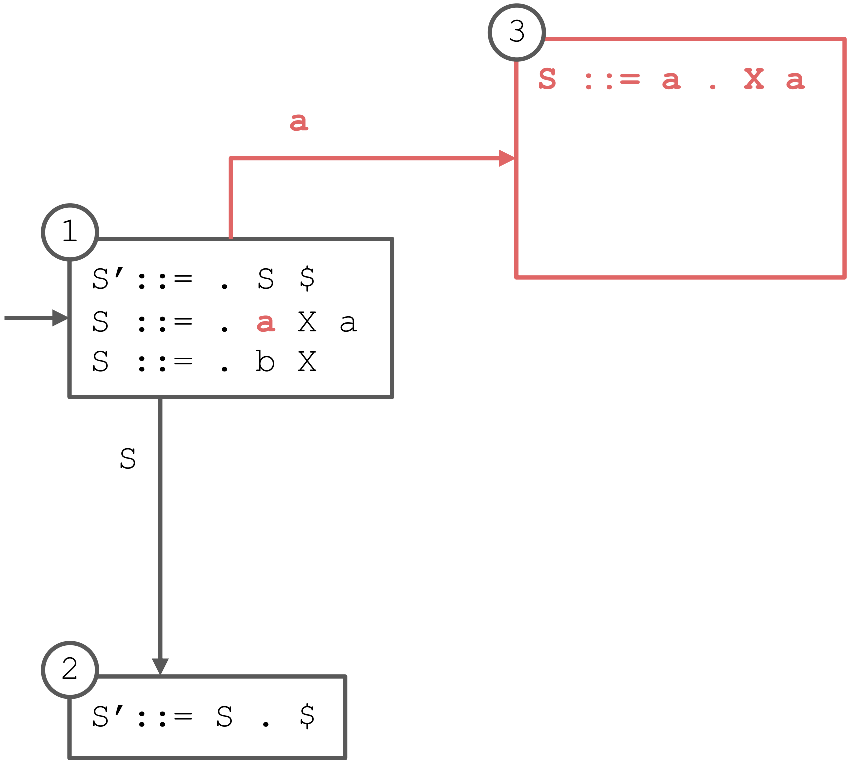


```
S’::= . S $
S ::= . a X a
S ::= . b X
```

```
S’::= S . $
```

State Diagram Construction

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



3

S ::= a . X a

a

1

S’::= . S $
S ::= . a X a
S ::= . b X

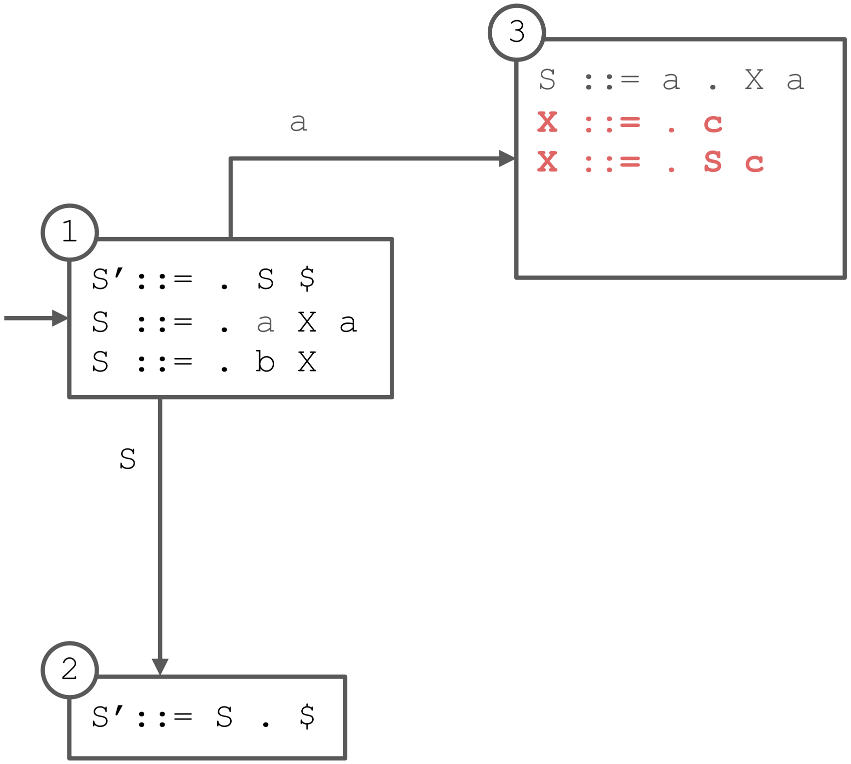
S

2

S’::= S . $

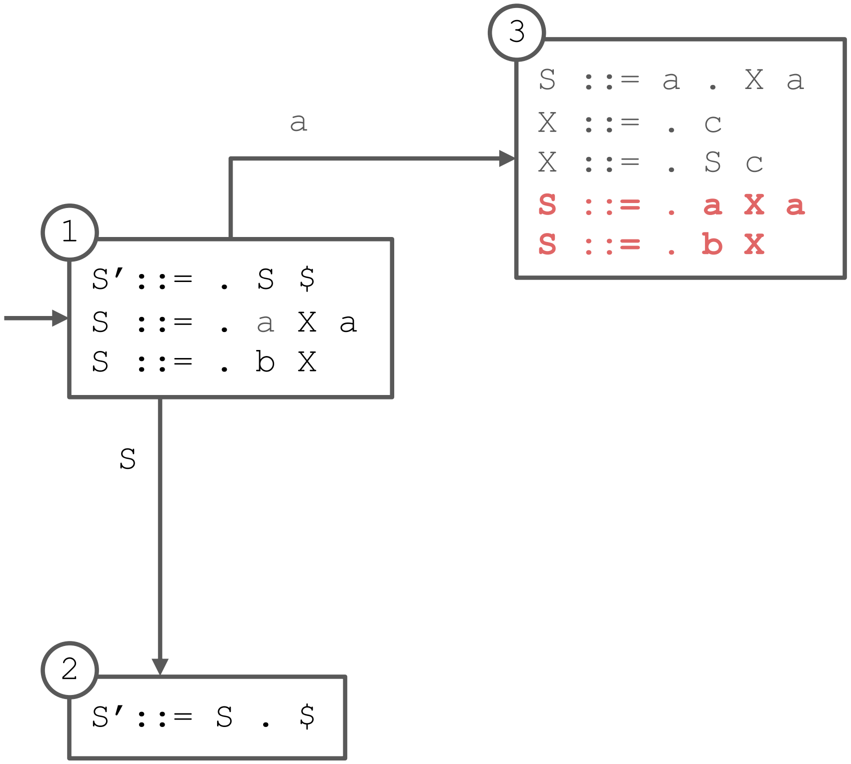
State Diagram Construction

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



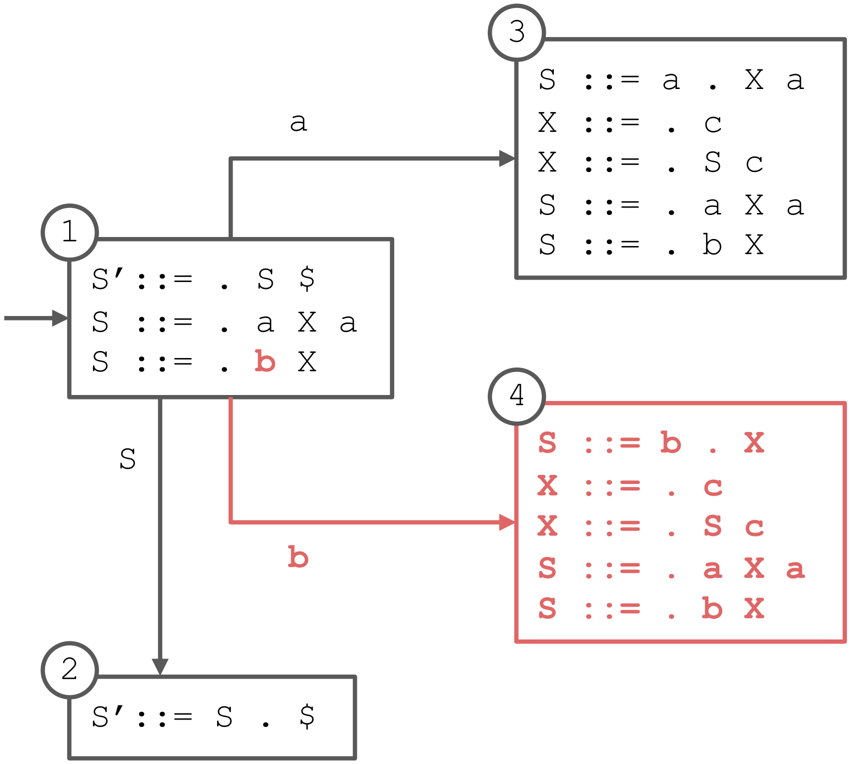
State Diagram Construction

```
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
```



```
1
S’::= . S $
S ::= . a X a
S ::= . b X
```

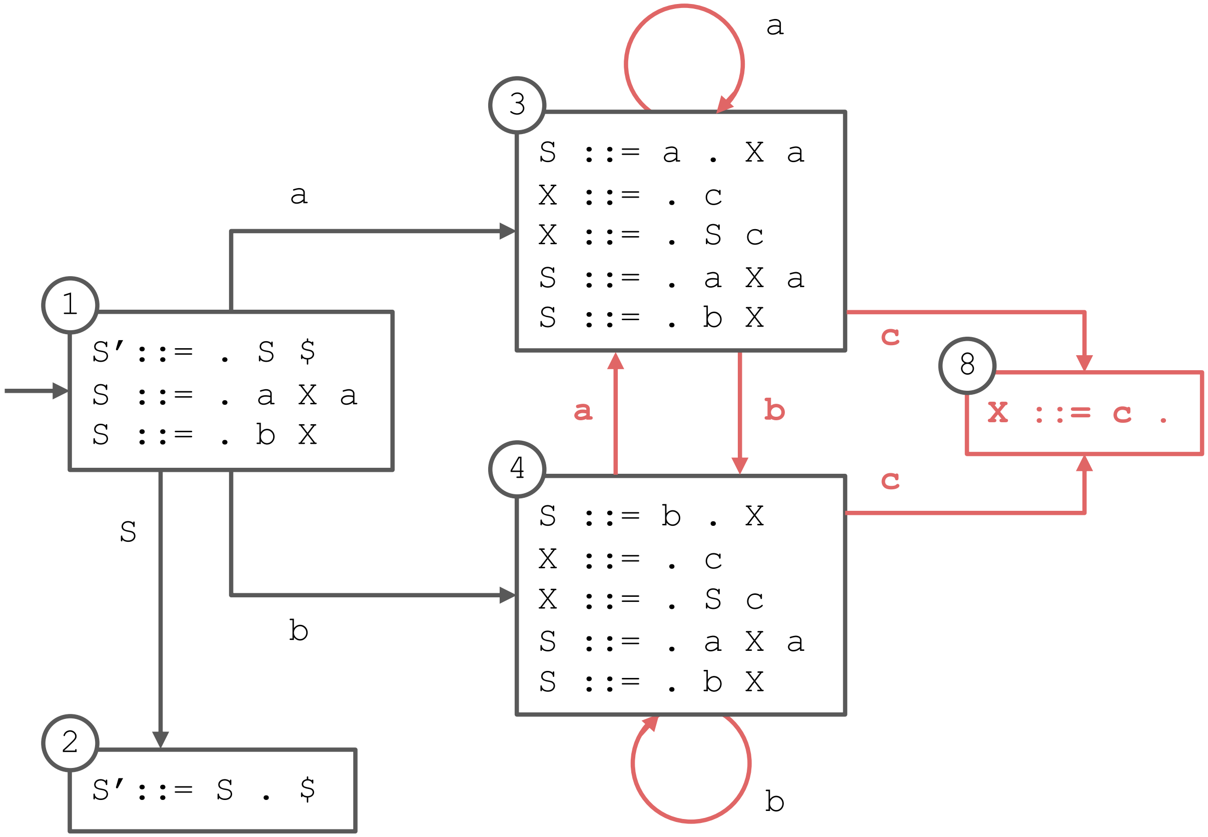
```
2
S’::= S . $
```

```
3
S ::= a . X a
X ::= . c
X ::= . S c
S ::= . a X a
S ::= . b X
```

```
4
S ::= b . X
X ::= . c
X ::= . S c
S ::= . a X a
S ::= . b X
```

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

State Diagram Construction



```
1
S’::= . S $
S ::= . a X a
S ::= . b X
```

```
2
S’::= S . $
```

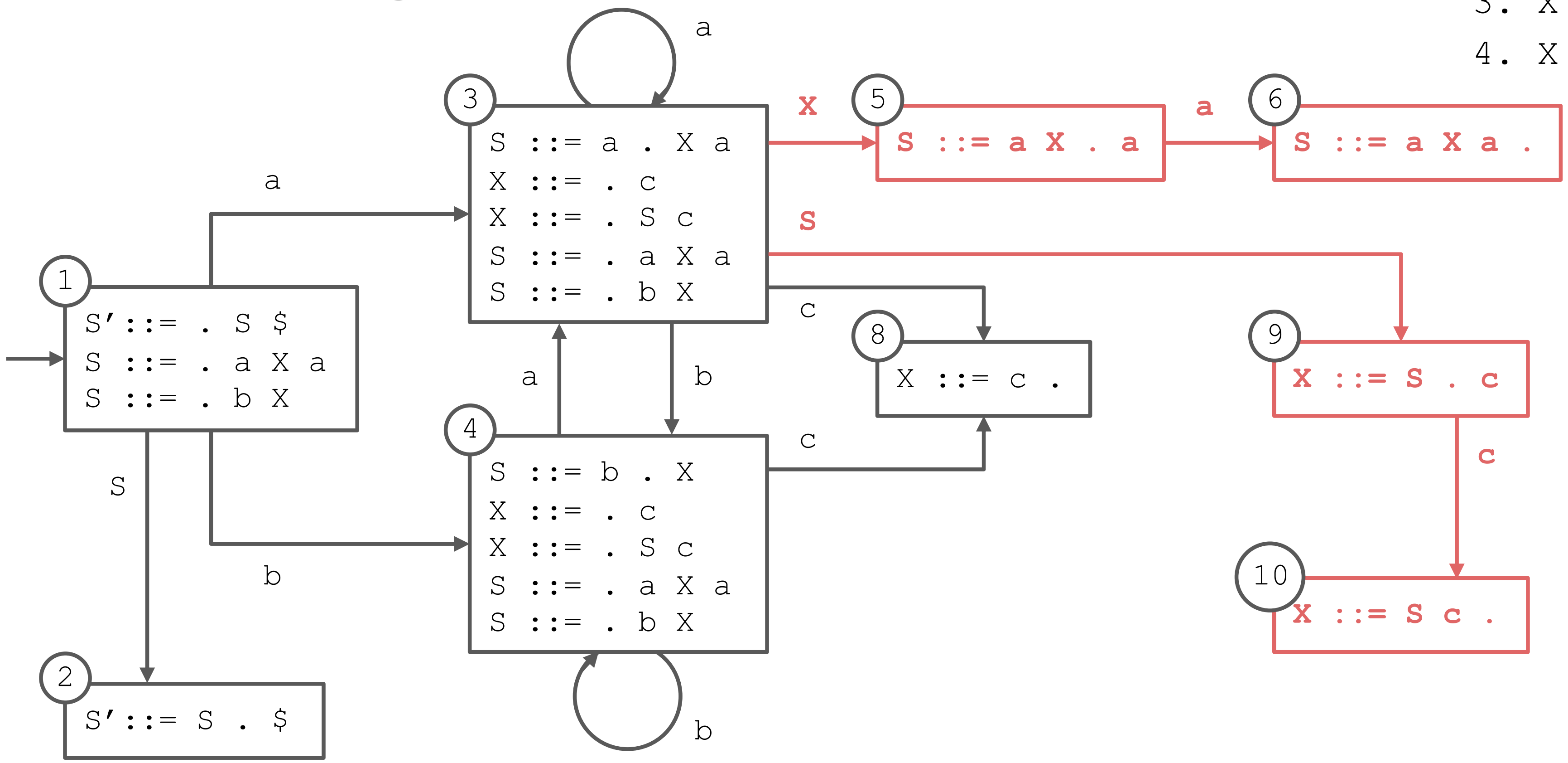
```
3
S ::= a . X a
X ::= . c
X ::= . S c
S ::= . a X a
S ::= . b X
```

```
4
S ::= b . X
X ::= . c
X ::= . S c
S ::= . a X a
S ::= . b X
```

```
8
X ::= c .
```

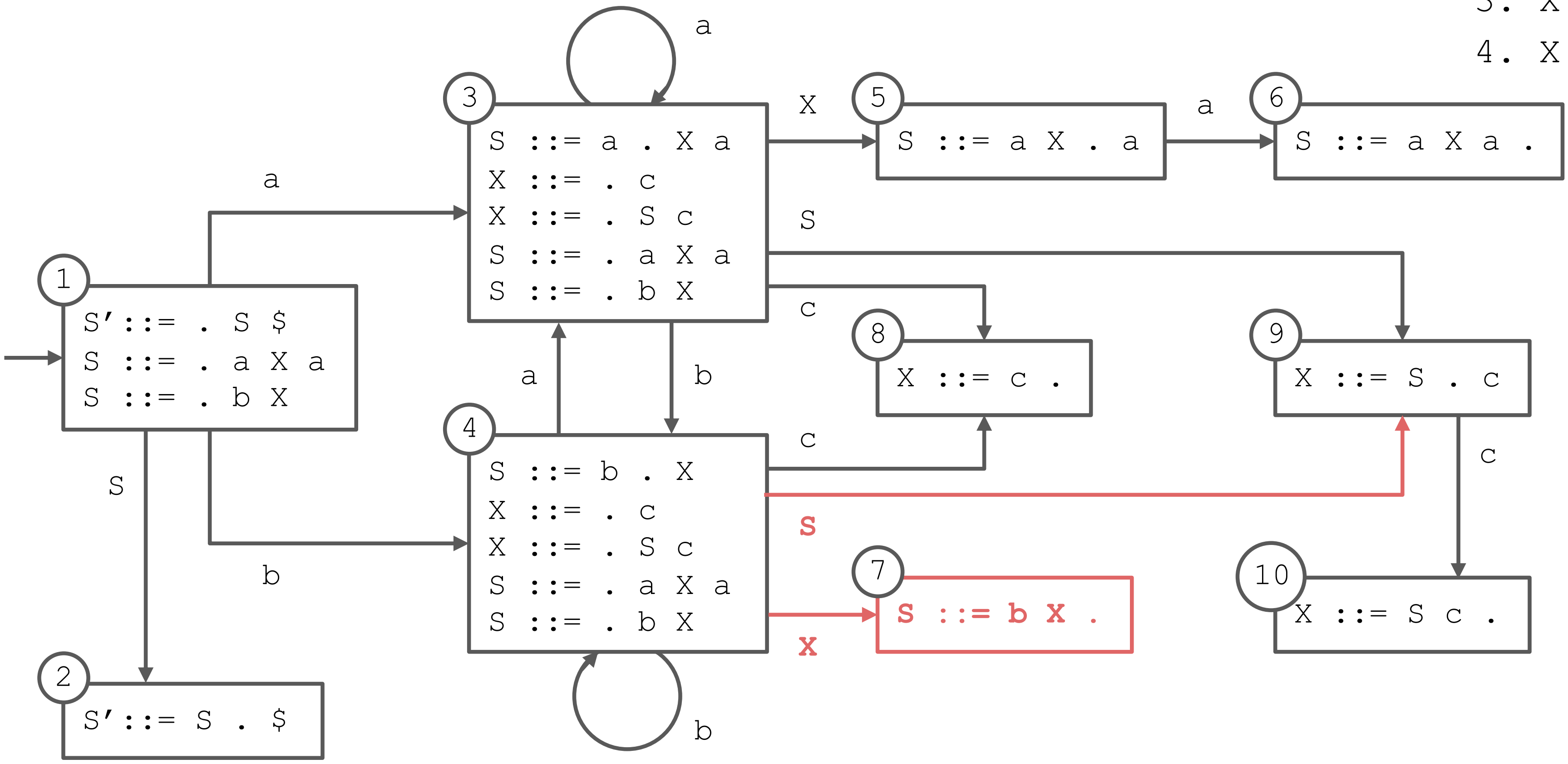
State Diagram Construction

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



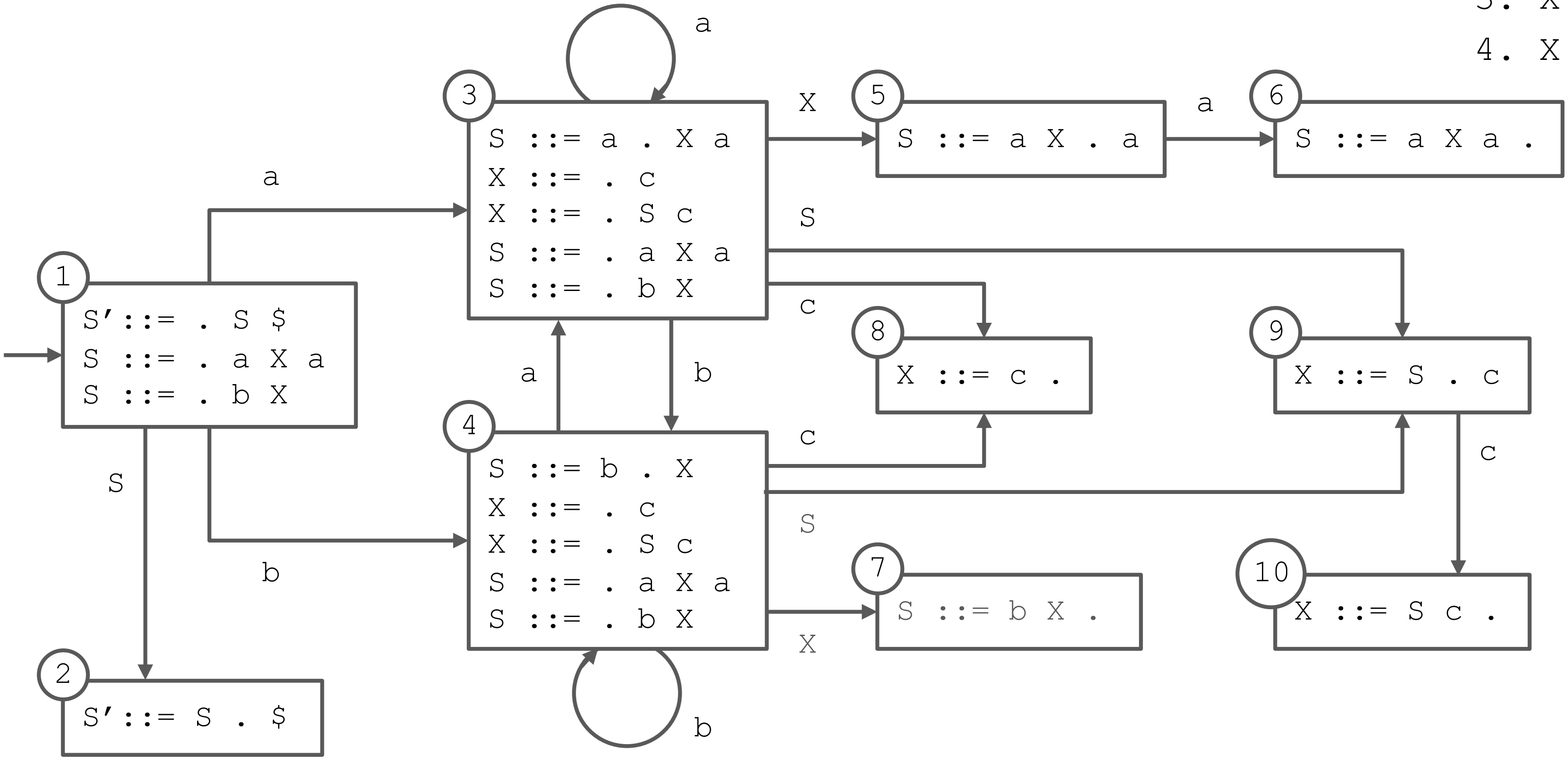
State Diagram Construction

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



Completed State Diagram

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



```
1
S’::= . S $
S ::= . a X a
S ::= . b X

2
S’::= S . $

3
S ::= a . X a
X ::= . c
X ::= . S c
S ::= . a X a
S ::= . b X

4
S ::= b . X
X ::= . c
X ::= . S c
S ::= . a X a
S ::= . b X

5
S ::= a X . a

6
S ::= a X a .

7
S ::= b X .

8
X ::= c .

9
X ::= S . c

10
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”

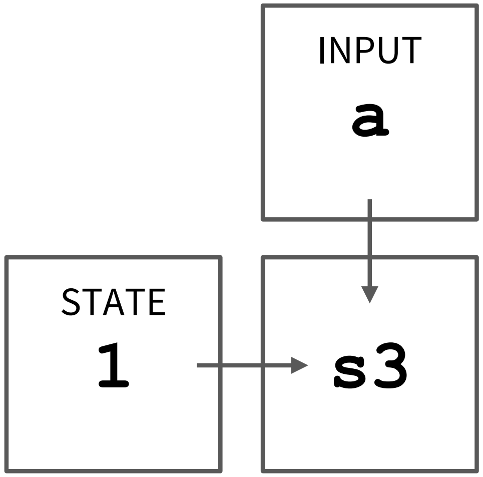
- occurs when the start symbol (S here) has been completed and there is no more input

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	r1	r1	r1	r1		
7	r2	r2	r2	r2		
8	r3	r3	r3	r3		
9			s10			
10	r4	r4	r4	r4		

Parse Trace

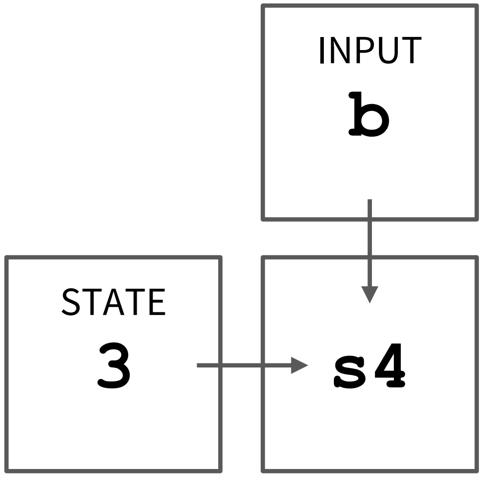
STACK	INPUT	ACTION
$ 1	a b c c a $	

Parse Trace



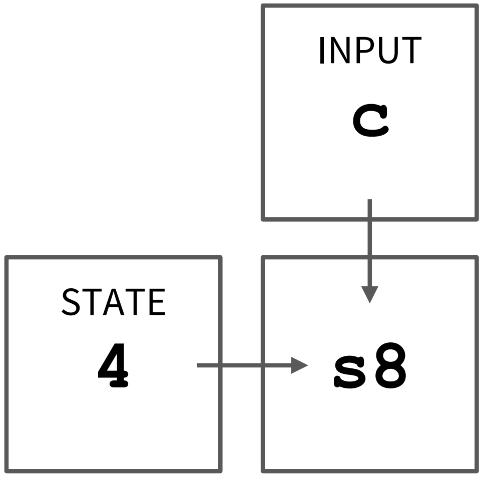
STACK	INPUT	ACTION
$ 1	a b c c a $	SHIFT
$ 1 a 3	b c c a $	

Parse Trace



STACK	INPUT	ACTION
$ 1	**a b c c a $**	SHIFT
$ 1 a 3	b c c a $	**SHIFT**
$ 1 a 3 b 4	**c c a $**	

Parse Trace



INPUT

c

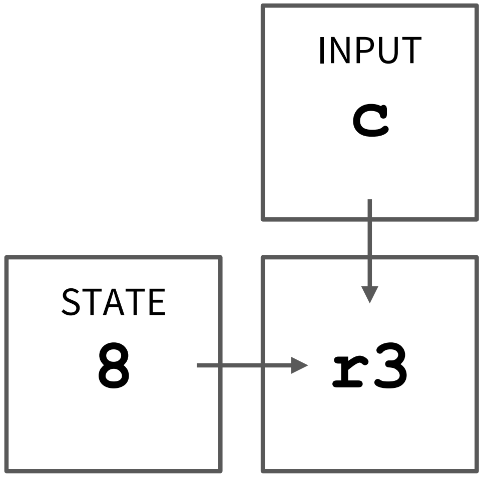
STATE

4

s8

STACK	INPUT	ACTION
$ 1	**a b c c a $**	SHIFT
$ 1 a 3	b c c a $	SHIFT
$ 1 a 3 b 4	c c a $	**SHIFT**
$ 1 a 3 b 4 c 8	**c a $**	

Parse Trace



INPUT

c

STATE

8

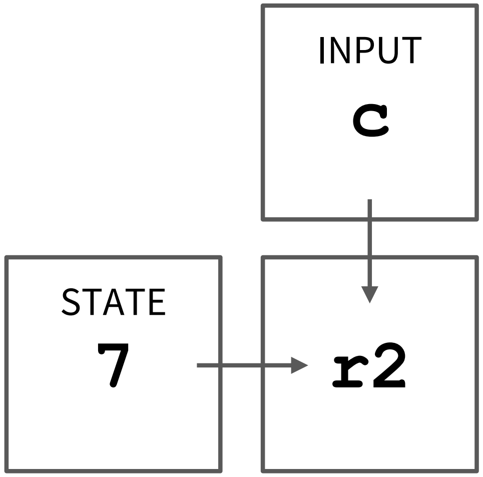
r3

3. X ::= c

(and GOTO step:
s4 & X -> g7)

STACK	INPUT	ACTION
$ 1	a b c c a $	SHIFT
$ 1 a 3	b c c a $	SHIFT
$ 1 a 3 b 4	c c a $	SHIFT
$ 1 a 3 b 4 c 8	c a $	REDUCE
$ 1 a 3 b 4 X 7	c a $	

Parse Trace

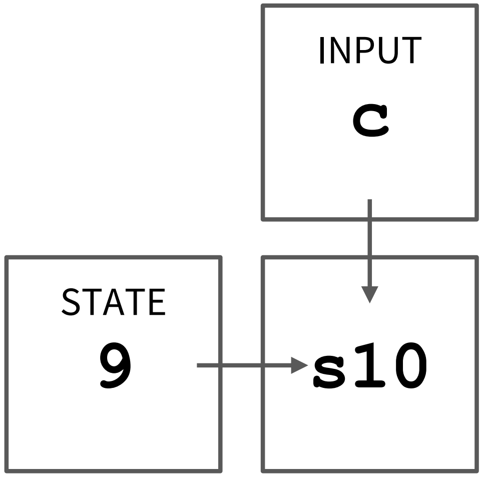


2. S ::= b X

(and GOTO step:
s3 & S -> g9)

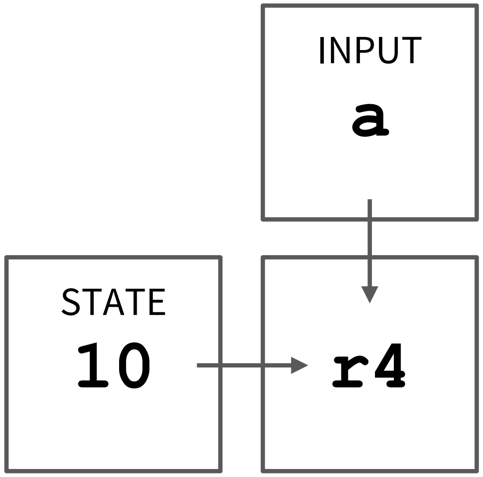
STACK	INPUT	ACTION
$ 1	**a b c c a $**	SHIFT
$ 1 a 3	b c c a $	SHIFT
$ 1 a 3 b 4	c c a $	SHIFT
$ 1 a 3 b 4 c 8	c a $	REDUCE
$ 1 a 3 b 4 X 7	c a $	**REDUCE**
$ 1 a 3 S 9	**c a $**	

Parse Trace



STACK	INPUT	ACTION
$ 1	**a b c c a $**	SHIFT
$ 1 a 3	b c c a $	SHIFT
$ 1 a 3 b 4	c c a $	SHIFT
$ 1 a 3 b 4 c 8	c a $	REDUCE
$ 1 a 3 b 4 X 7	c a $	REDUCE
$ 1 a 3 S 9	c a $	**SHIFT**
$ 1 a 3 S 9 c 10	**a $**	

Parse Trace


INPUT

a

STATE

10

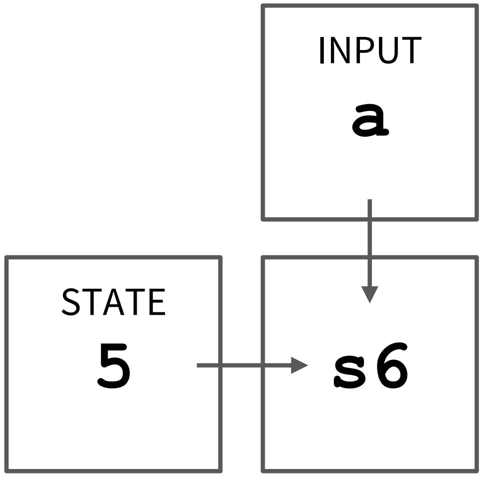
r4

4. X ::= S c

(and GOTO step:
s3 & X -> g5)

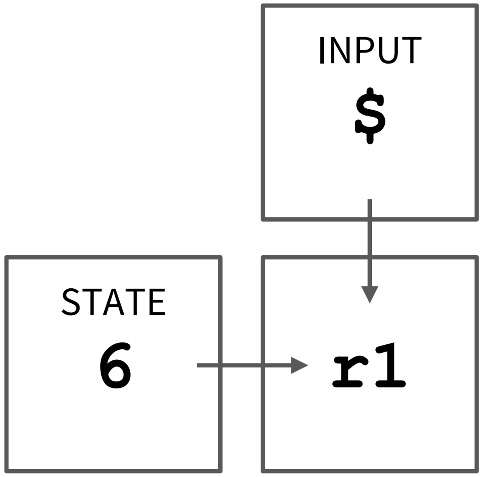
STACK	INPUT	ACTION
$ 1	**a b c c a $**	SHIFT
$ 1 a 3	b c c a $	SHIFT
$ 1 a 3 b 4	c c a $	SHIFT
$ 1 a 3 b 4 c 8	c a $	REDUCE
$ 1 a 3 b 4 X 7	c a $	REDUCE
$ 1 a 3 S 9	c a $	SHIFT
$ 1 a 3 S 9 c 10	a $	**REDUCE**
$ 1 a 3 X 5	**a $**	

Parse Trace



STACK	INPUT	ACTION
$ 1	**a b c c a $**	SHIFT
$ 1 a 3	b c c a $	SHIFT
$ 1 a 3 b 4	c c a $	SHIFT
$ 1 a 3 b 4 c 8	c a $	REDUCE
$ 1 a 3 b 4 X 7	c a $	REDUCE
$ 1 a 3 S 9	c a $	SHIFT
$ 1 a 3 S 9 c 10	a $	REDUCE
$ 1 a 3 X 5	a $	**SHIFT**
$ 1 a 3 X 5 a 6	**$**	

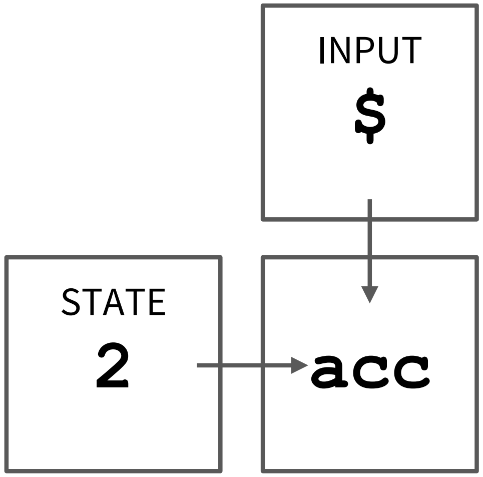
Parse Trace



```
1. S ::= a X a
(and GOTO step:
 s1 & S -> g2)
```

STACK	INPUT	ACTION
$ 1	**a b c c a $**	SHIFT
$ 1 a 3	b c c a $	SHIFT
$ 1 a 3 b 4	c c a $	SHIFT
$ 1 a 3 b 4 c 8	c a $	REDUCE
$ 1 a 3 b 4 X 7	c a $	REDUCE
$ 1 a 3 S 9	c a $	SHIFT
$ 1 a 3 S 9 c 10	a $	REDUCE
$ 1 a 3 X 5	a $	SHIFT
$ 1 a 3 X 5 a 6	$	**REDUCE**
$ 1 S 2	**$**	

Parse Trace



INPUT

$

STATE

2

acc

STACK	INPUT	ACTION
$ 1	**a b c c a $**	SHIFT
$ 1 a 3	b c c a $	SHIFT
$ 1 a 3 b 4	c c a $	SHIFT
$ 1 a 3 b 4 c 8	c a $	REDUCE
$ 1 a 3 b 4 X 7	c a $	REDUCE
$ 1 a 3 S 9	c a $	SHIFT
$ 1 a 3 S 9 c 10	a $	REDUCE
$ 1 a 3 X 5	a $	SHIFT
$ 1 a 3 X 5 a 6	$	REDUCE
$ 1 S 2	$	ACCEPT