Chapter 22

Bottom-up Parsing

Bottom-up parsing

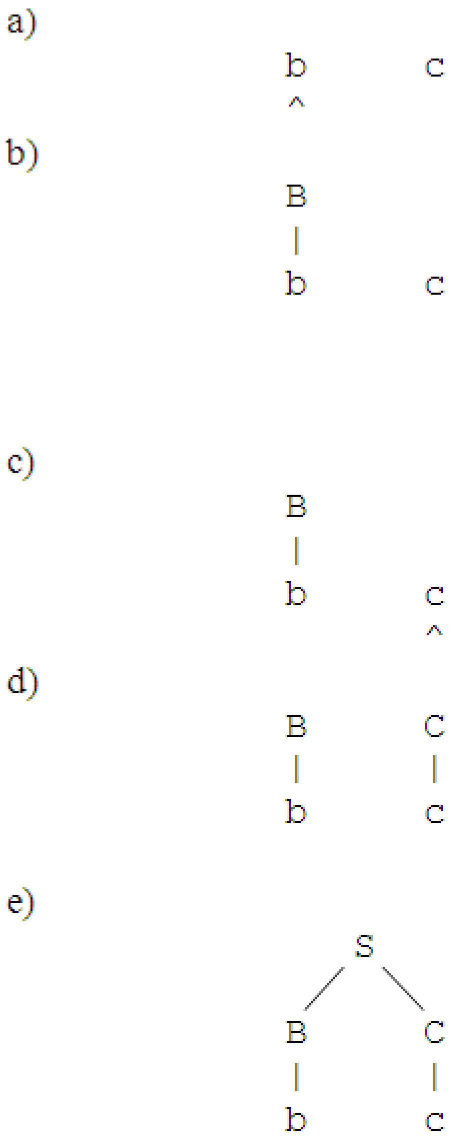
Construct the parse starting at the bottom, ending with the start symbol

Example of a bottom-up parse

```
S  ->  BC
B  ->  b
C  ->  c
```

Parse `bc`

Parse `bc`



Uncovers rightmost derivation

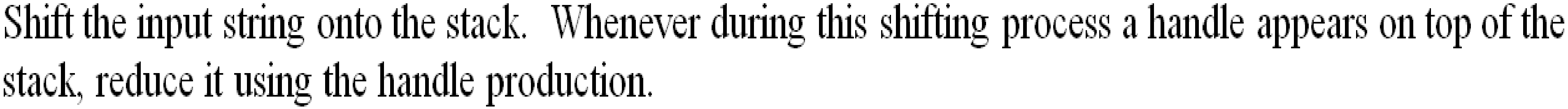
S => BC => Bc => bc

Reduce operation

Handle: Symbols on top of the stack that make up the right side of a production (the handle production)used in a rightmost derivation.

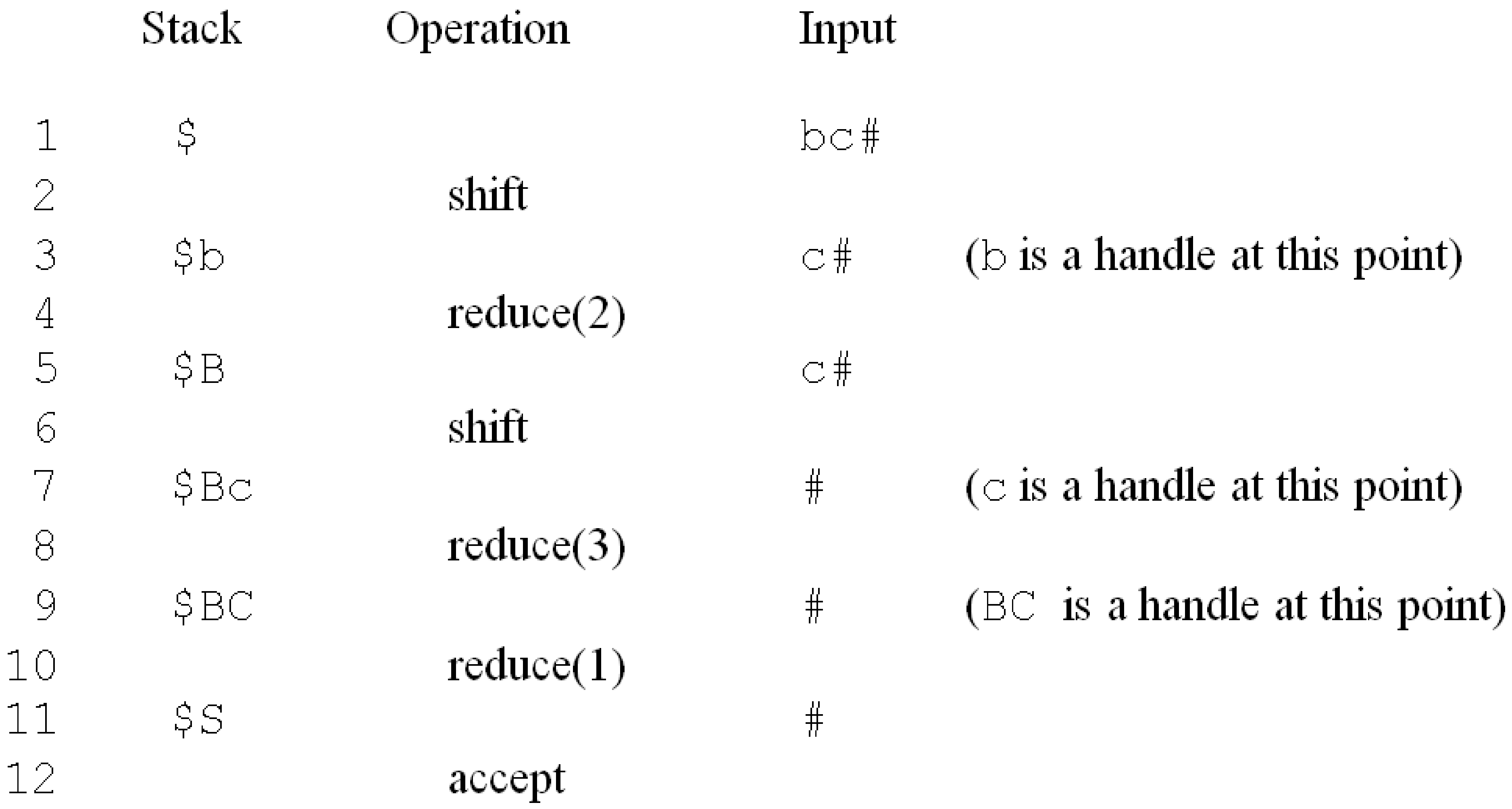
Reduce operation: replacing handle with the left side of the handle production.

When to reduce



Shift the input string onto the stack. Whenever during this shifting process a handle appears on top of the stack, reduce it using the handle production.

Using a stack



	Stack	Operation	Input	
1	$		bc#	
2		shift		
3	$b		c#	(b is a handle at this point)
4		reduce(2)		
5	$B		c#	
6		shift		
7	$Bc		#	(c is a handle at this point)
8		reduce(3)		
9	$BC		#	(BC is a handle at this point)
10		reduce(1)		
11	$S		#	
12		accept		

Right side not always a handle

1) S → cS
2) S → c

Using this grammar, let's parse cc (See Fig. 22.3).

1	$		cc#	
2		shift		
3	$c		c#	(c on stack is not a handle)
4		shift		
5	#cc		#	
6		reduce(2)		
7	#cS		#	
8		reduce(1)		
9	#S		#	
10		accept		

Using left recursive productions

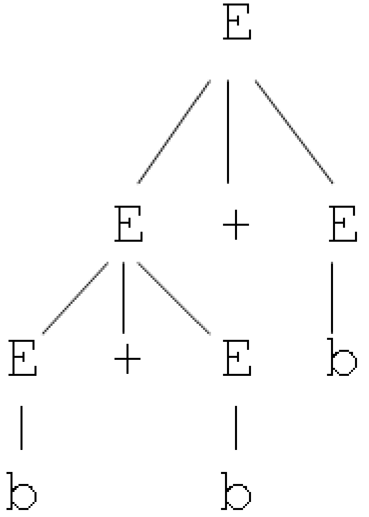
1) S → Sc
2) S → c

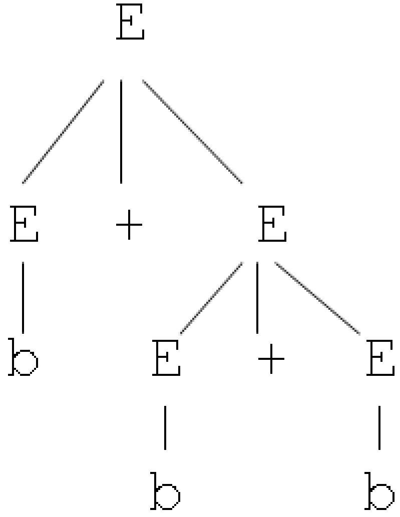
1	$		cc#	
2		shift		
3	$c		c#	(c on stack is a handle)
4		reduce(2)		
5	#S		c#	
6		shift		
7	#Sc		#	
8		reduce(2)		
9	#S		#	
10		accept		

Stack size never more than 2

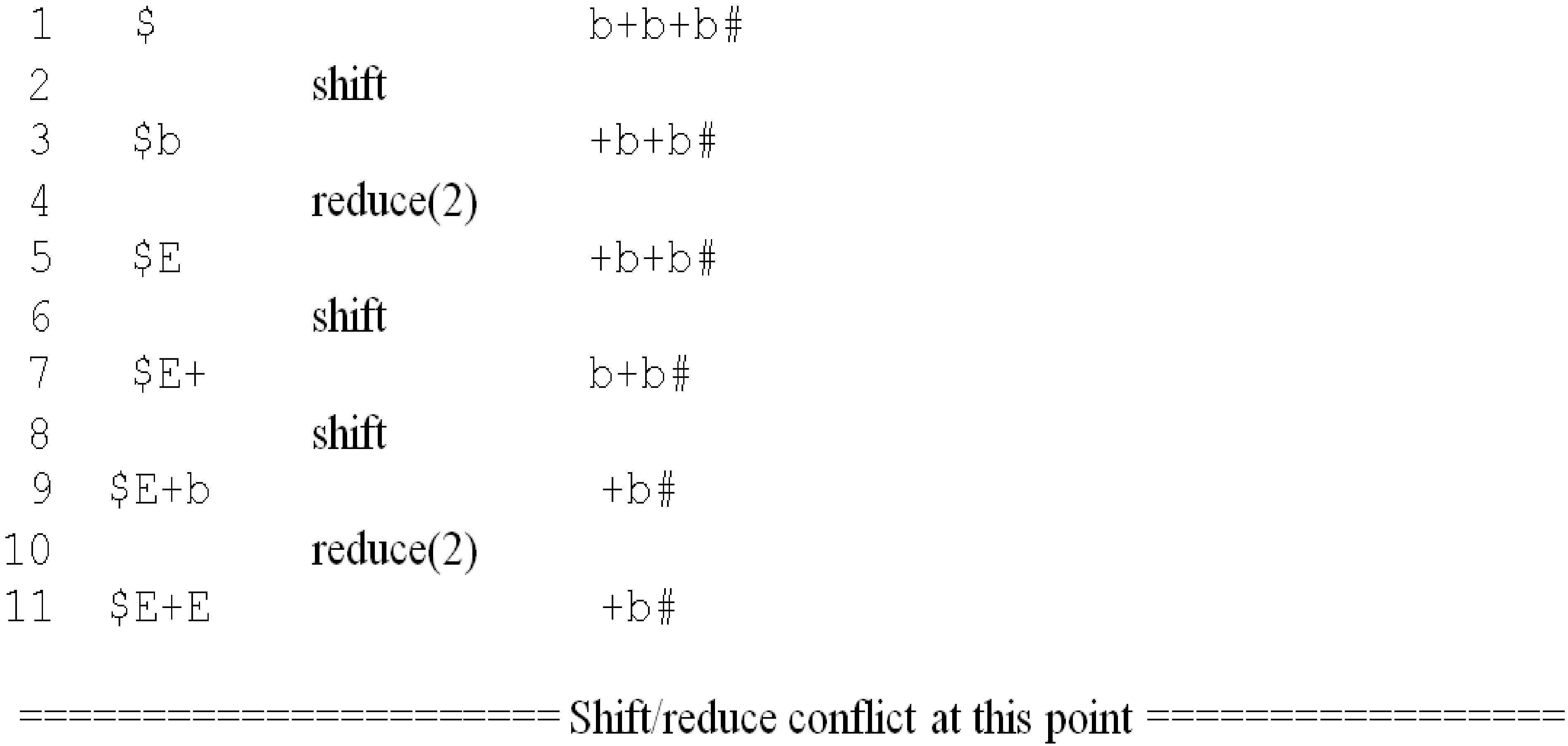
Using ambiguous grammars

1) E → E + E
2) E → b





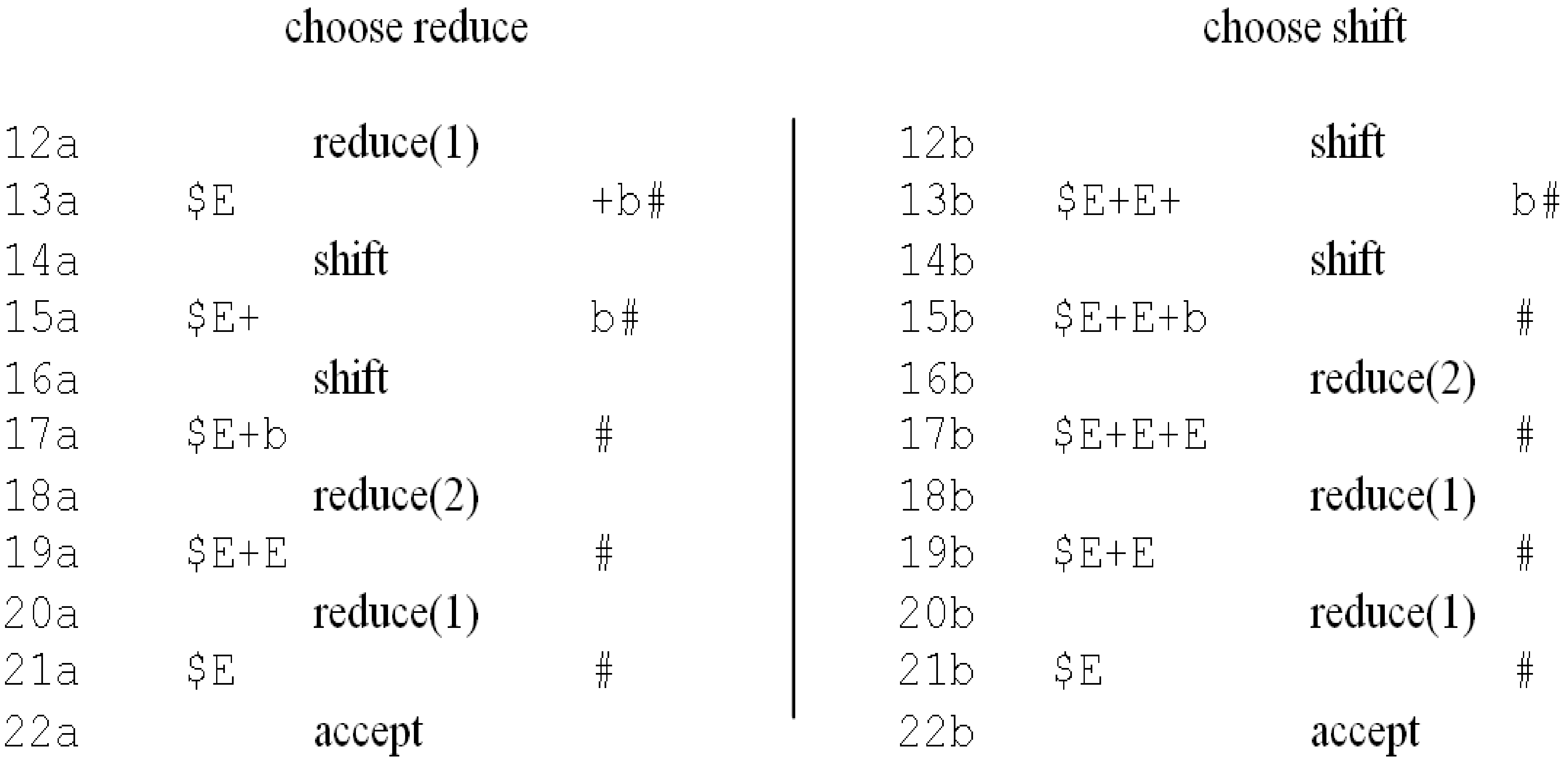
Shift/reduce conflict



```
 1    $                      b+b+b#
 2            shift
 3    $b                     +b+b#
 4            reduce(2)
 5    $E                     +b+b#
 6            shift
 7    $E+                    b+b#
 8            shift
 9   $E+b                     +b#
10            reduce(2)
11   $E+E                     +b#

===================== Shift/reduce conflict at this point ===================
```

Shift or reduce determines parse tree



		choose reduce				choose shift	
12a		reduce(1)		12b		shift	
13a	$E		+b#	13b	$E+E+		b#
14a		shift		14b		shift	
15a	$E+		b#	15b	$E+E+b		#
16a		shift		16b		reduce(2)	
17a	$E+b		#	17b	$E+E+E		#
18a		reduce(2)		18b		reduce(1)	
19a	$E+E		#	19b	$E+E		#
20a		reduce(1)		20b		reduce(1)	
21a	$E		#	21b	$E		#
22a		accept		22b		accept	

Effect of shift or reduce

Shift gives higher precedence to operator on the stack. Reduce gives higher precedence to operator that is the current token.

Do-not-reduce rule

Do not reduce by a production if the current input is not in the FOLLOW set of the production's left side.

```
S  ->  bS
S  ->  b
```

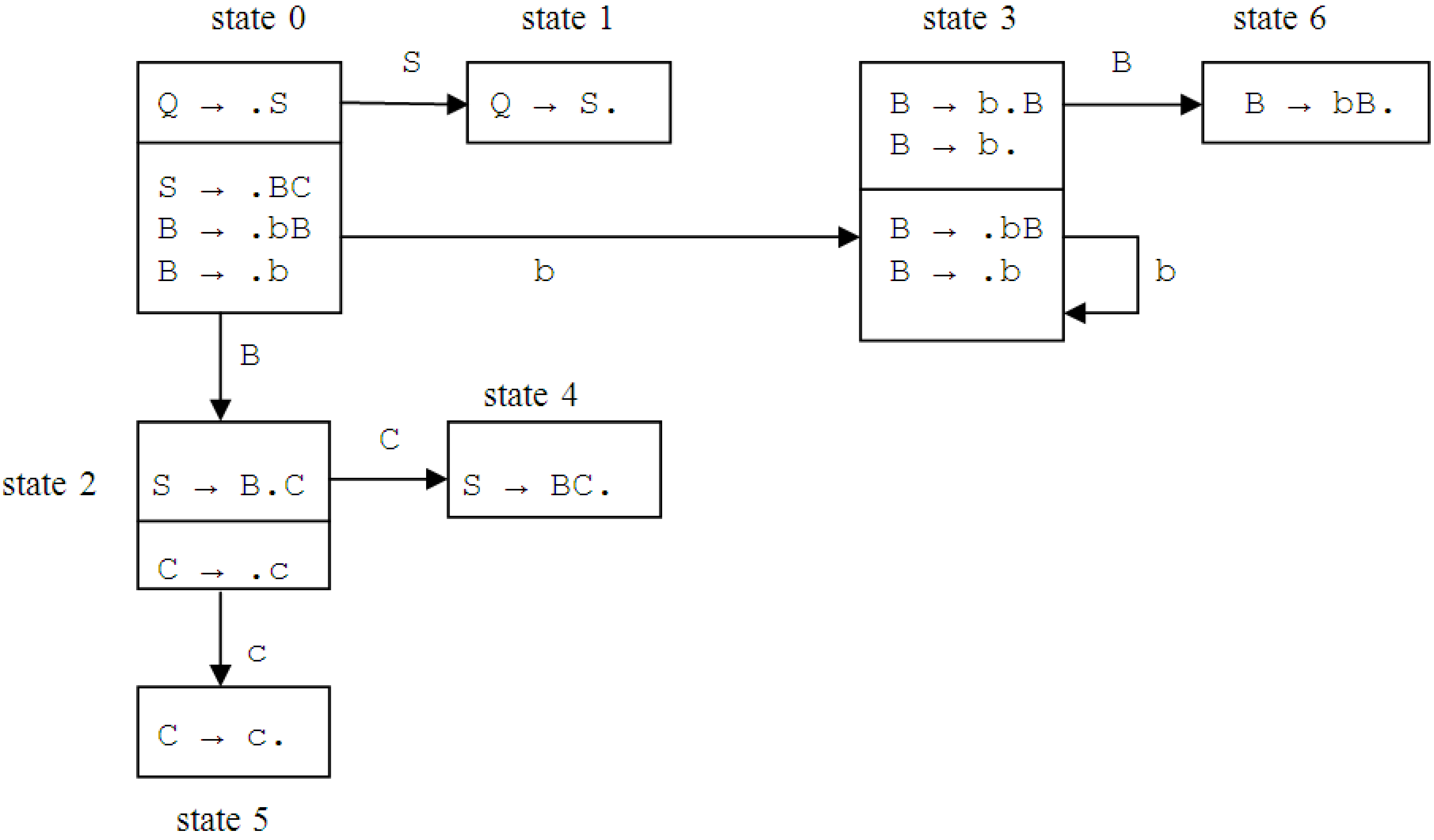
```
$                 bb#
        shift
$b                b#   (Do not reduce  here)
```

SLR(1) Parsing

1) S → BC
2) B → bB
3) B → b
4) C → c

0) Q → S
1) S → BC
2) B → bB
3) B → b
4) C → c

SLR(1) Parsing



state 0

Q → .S

S → .BC
B → .bB
B → .b

state 1

Q → S.

state 3

B → b.B
B → b.

B → .bB
B → .b

state 6

B → bB.

state 2

S → B.C

C → .c

state 4

S → BC.

state 5

C → c.

SLR(1)

		input			State to push when left side of reducing production is		
		b	c	#	S	B	C
	0	s3			1	2	
	1			accept			
	2		s5				4
state	3	s3	r3			6	
	4			r1			
	5			r4			
	6		r2				

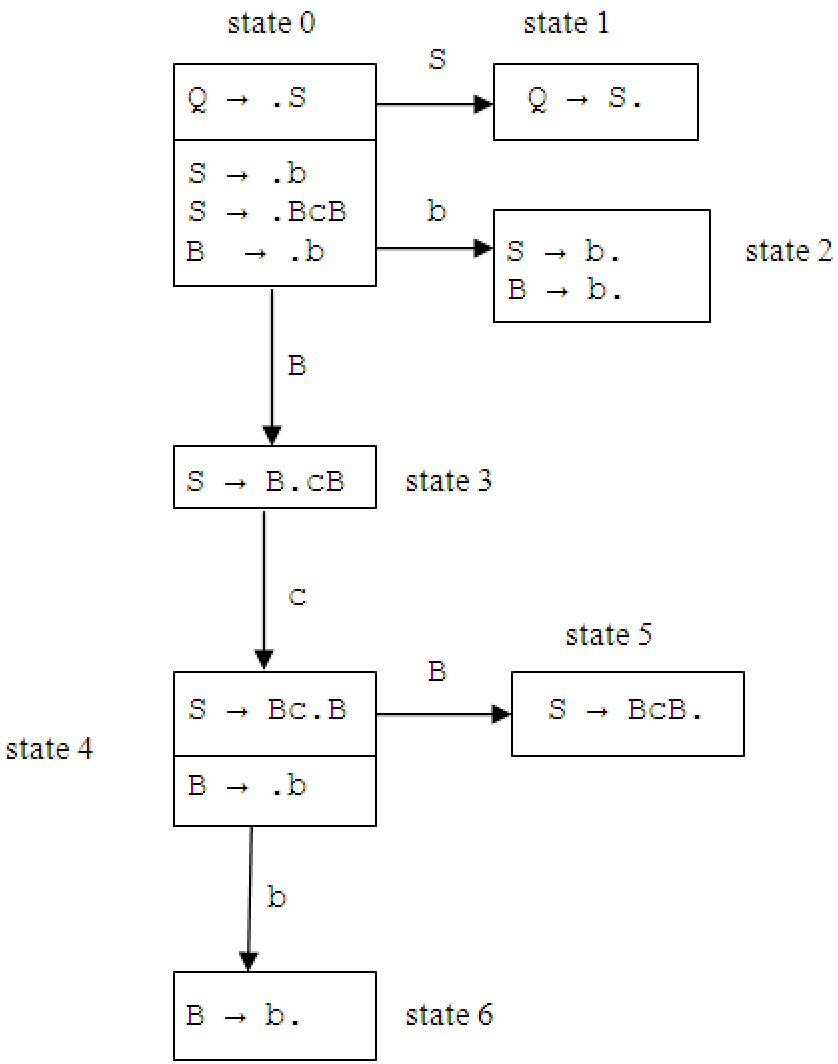
Parse using the table

$0	bbc#
$03	bc#
$033	c#
$036	c#
$02	c#
$024	#
$01	#

Shift/reduce conflicts

Shift to give operator on stack higher precedence.

Reduce/reduce conflict state 2



state 0

Q → .S

S → .b
S → .BcB
B → .b

state 1

Q → S.

state 2

S → b.
B → b.

state 3

S → B.cB

state 4

S → Bc.B

B → .b

state 5

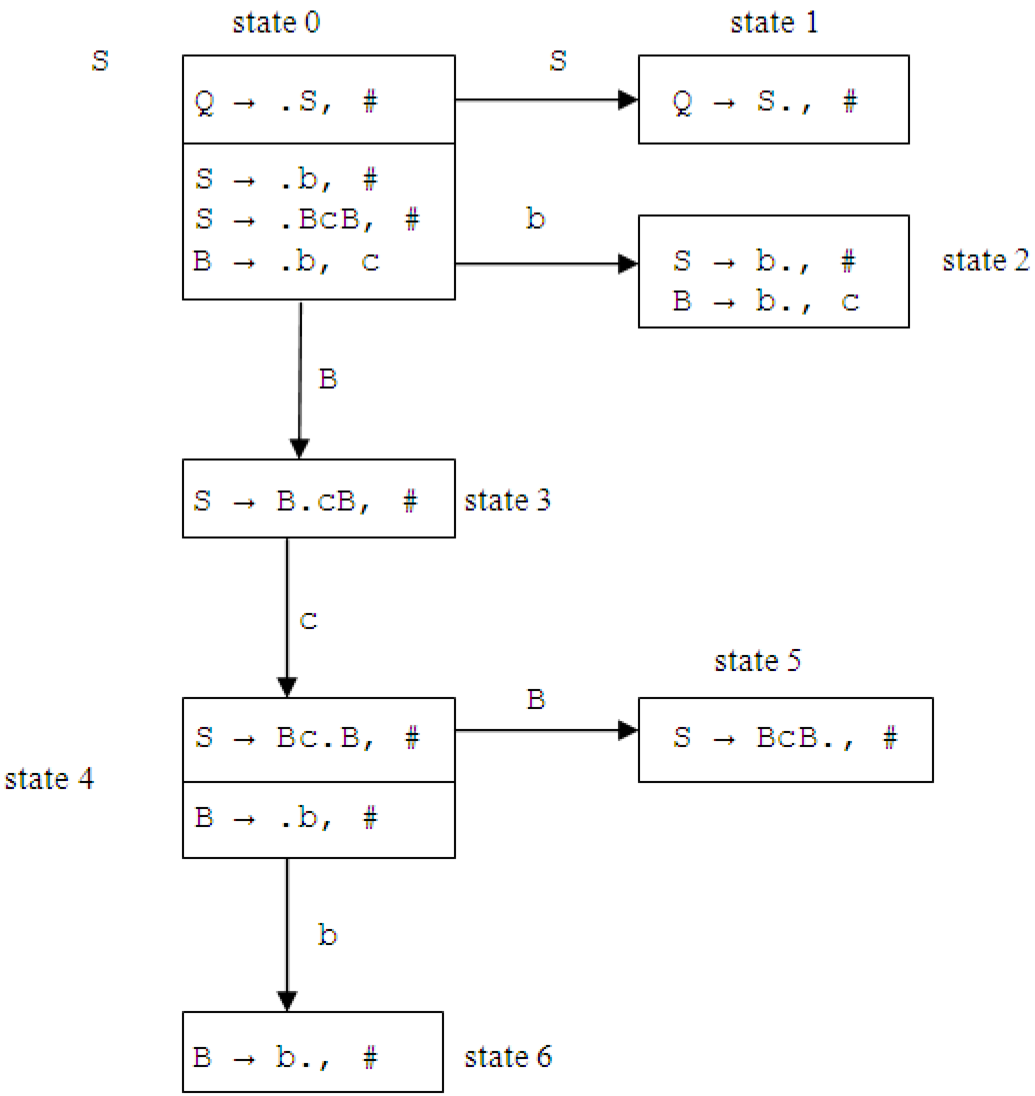
S → BcB.

state 6

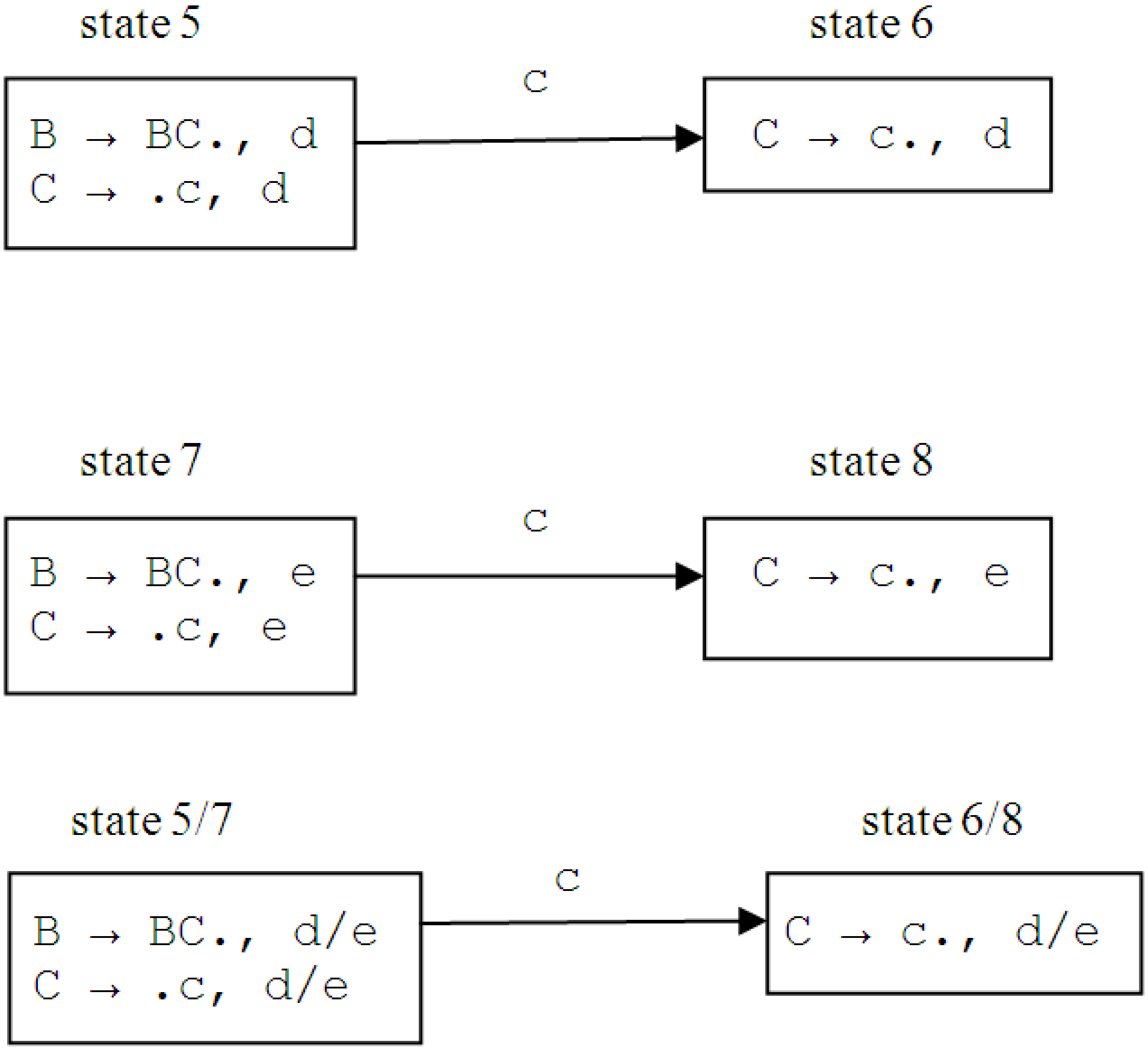
B → b.

			input		State t to push when left side of reducing production is	
		b	c	#	S	B
	0	s2			1	3
	1			accept		
state	2		r3	r1/r3		
	3		s4			
	4	s6				5
	5			r2		
	6		r3	r3		

LR(1) Parsing


state 0
S
Q → .S, #
S → .b, #
S → .BcB, #
B → .b, c
state 1
S
Q → S., #
b
S → b., #
B → b., c
state 2
B
S → B.cB, #
state 3
c
state 5
B
S → Bc.B, #
B → .b, #
S → BcB., #
state 4
b
B → b., #
state 6

LALR Parsing



state 5

B → BC., d
C → .c, d

c

state 6

C → c., d

state 7

B → BC., e
C → .c, e

c

state 8

C → c., e

state 5/7

B → BC., d/e
C → .c, d/e

c

state 6/8

C → c., d/e