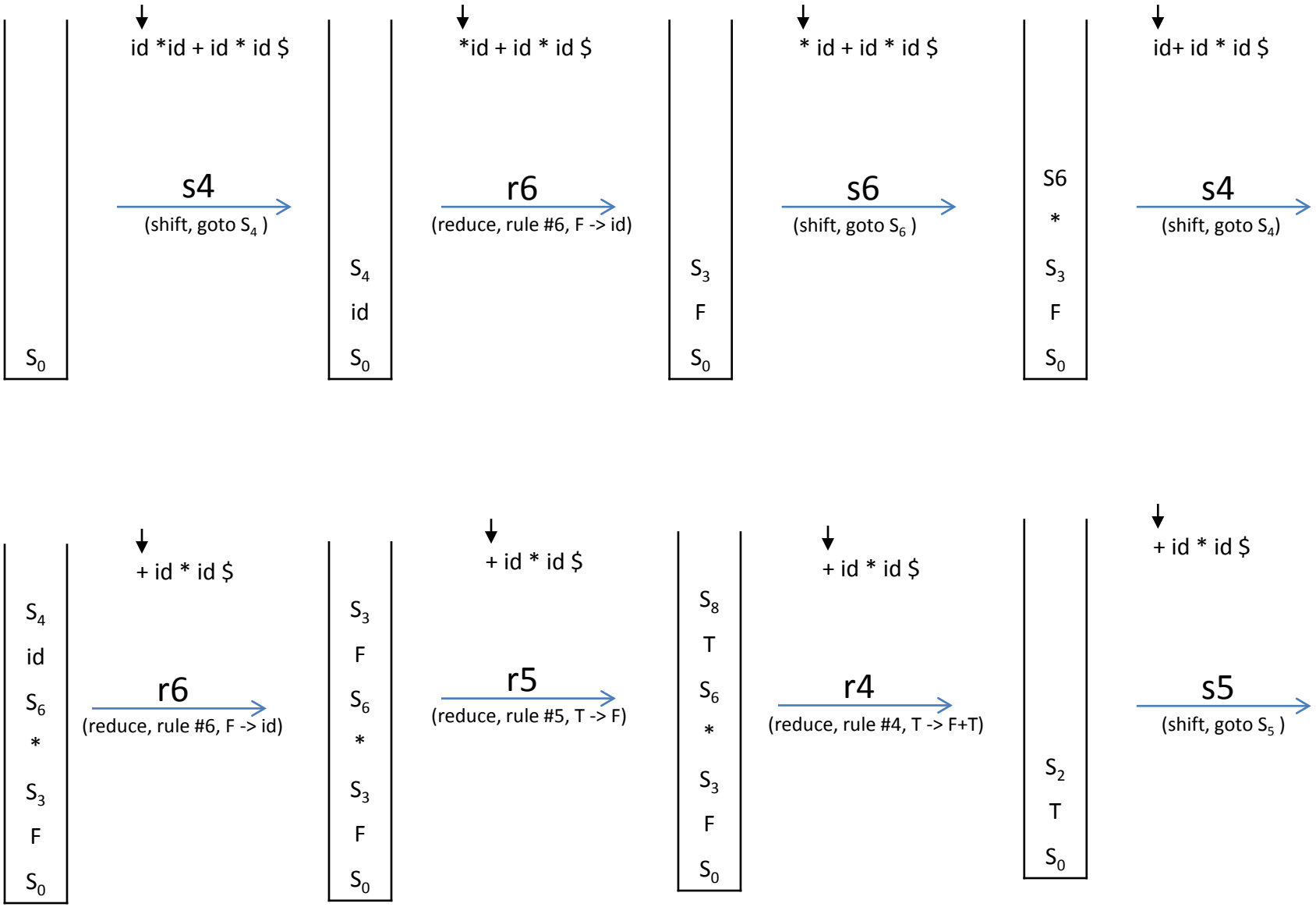
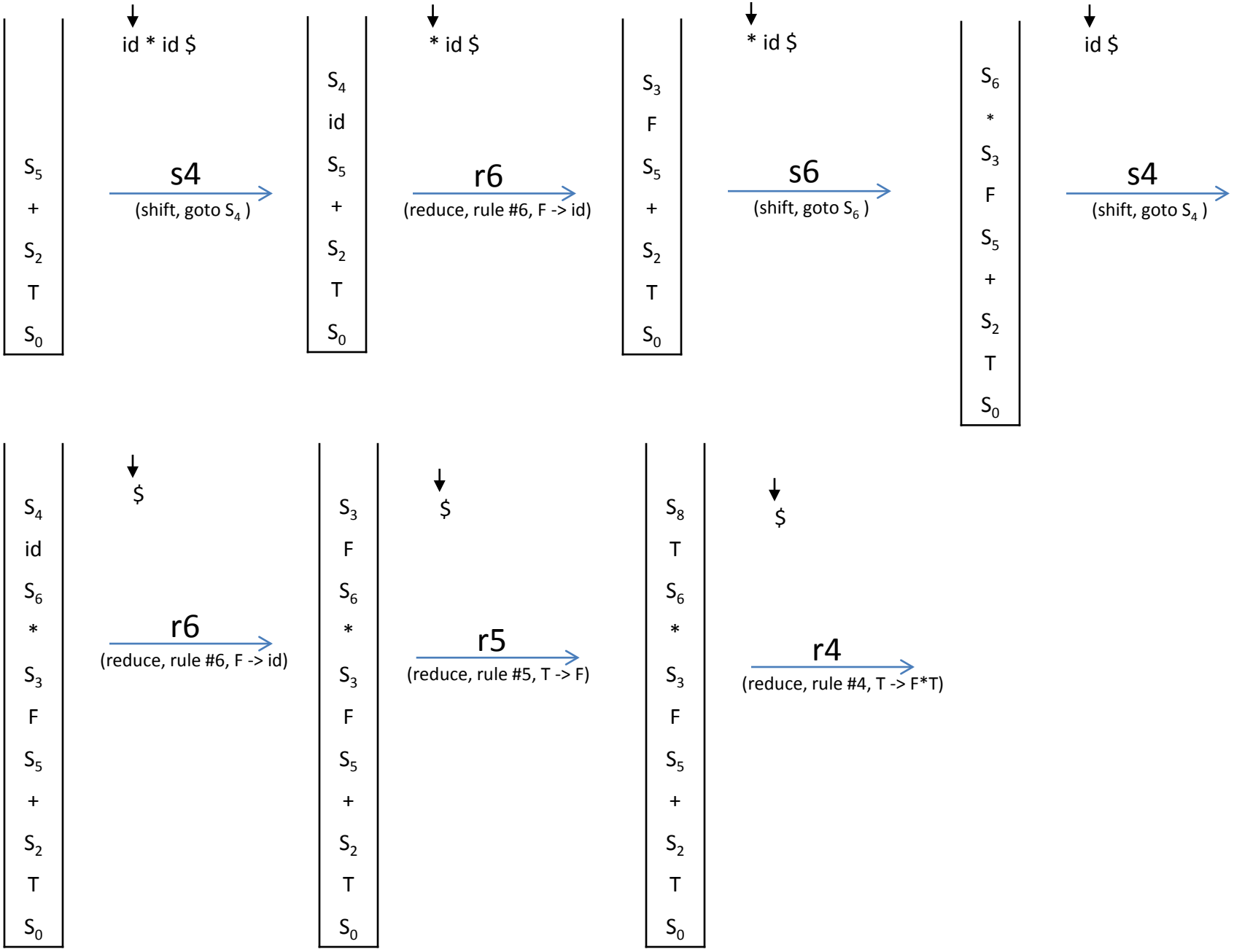
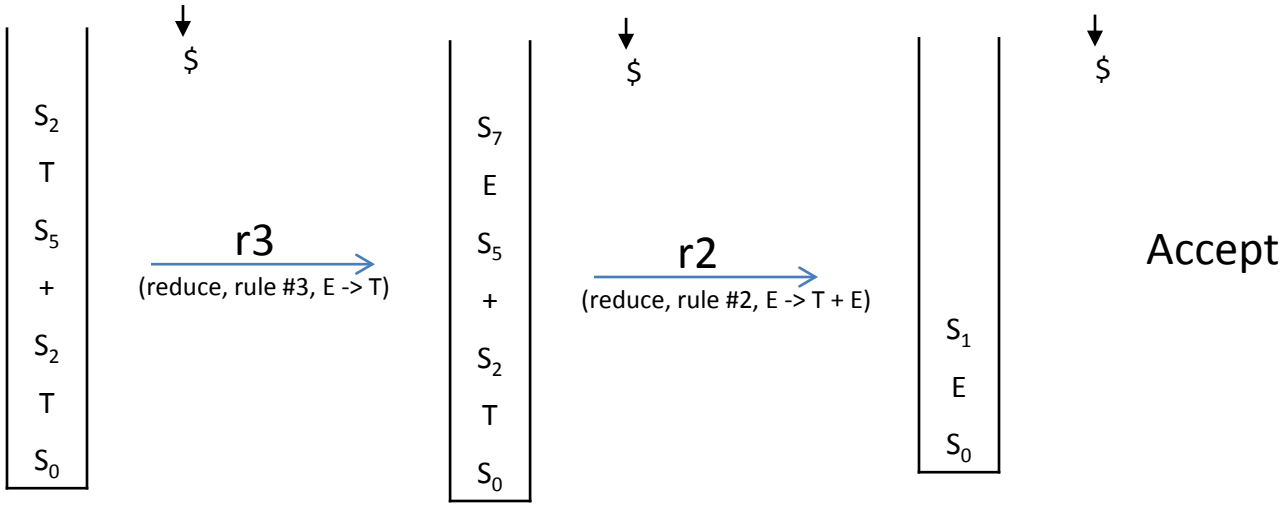
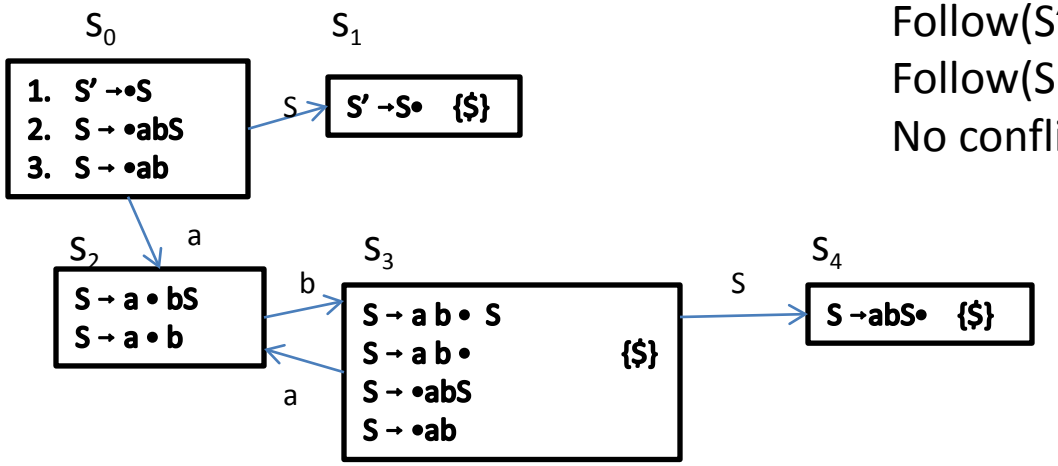
1. Parsing id*id+id*id




S_5 + S_2 T S_0
id * id $
s4
(shift, goto S_4)
S_4 id S_5 + S_2 T S_0
* id $
r6
(reduce, rule #6, F -> id)
S_3 F S_5 + S_2 T S_0
* id $
s6
(shift, goto S_6)
S_6 * S_3 F S_5 + S_2 T S_0
id $
s4
(shift, goto S_4)
S_4 id S_6 * S_3 F S_5 + S_2 T S_0
$
r6
(reduce, rule #6, F -> id)
S_3 F S_6 * S_3 F S_5 + S_2 T S_0
$
r5
(reduce, rule #5, T -> F)
S_8 T S_6 * S_3 F S_5 + S_2 T S_0
$
r4
(reduce, rule #4, T -> F*T)


$
S2
T
S5
+
S2
T
S0
r3
(reduce, rule #3, E -> T)
$
S7
E
S5
+
S2
T
S0
r2
(reduce, rule #2, E -> T + E)
$
S1
E
S0
Accept

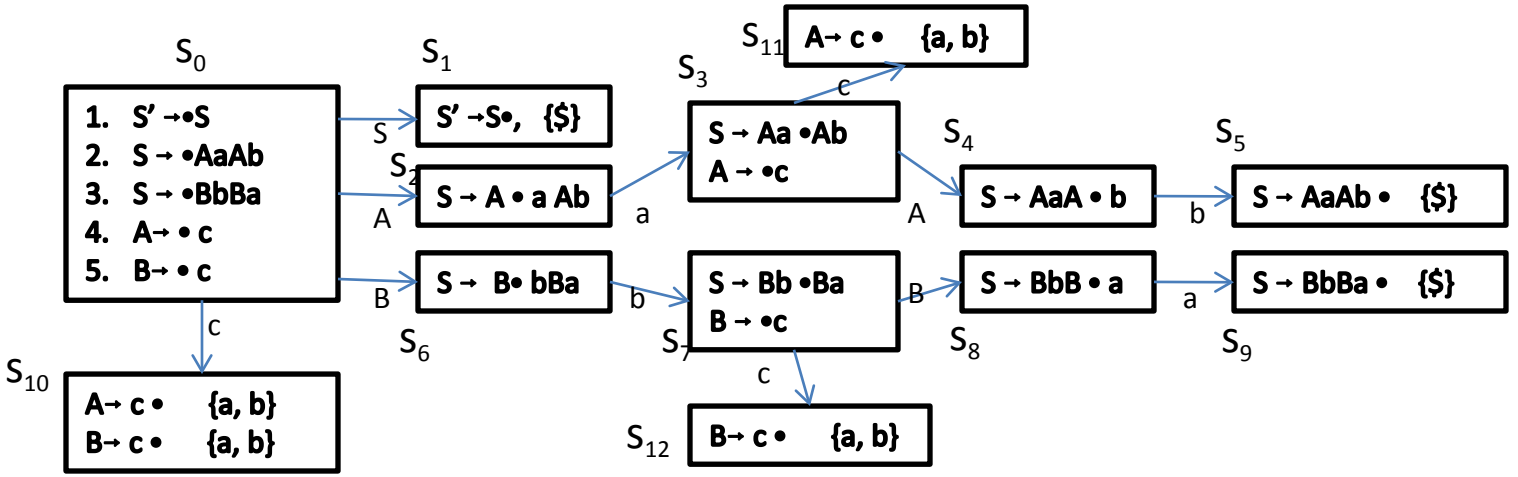
2. (a)



Follow(S') = {$}
Follow(S) = {$}
No conflict. It is SLR.

	Action			GOTO
	a	b	$	S
S_0	s2			1
S_1			!	
S_2		s3		
S_3	s2		r3	4
S_4			r2	

2. (b)

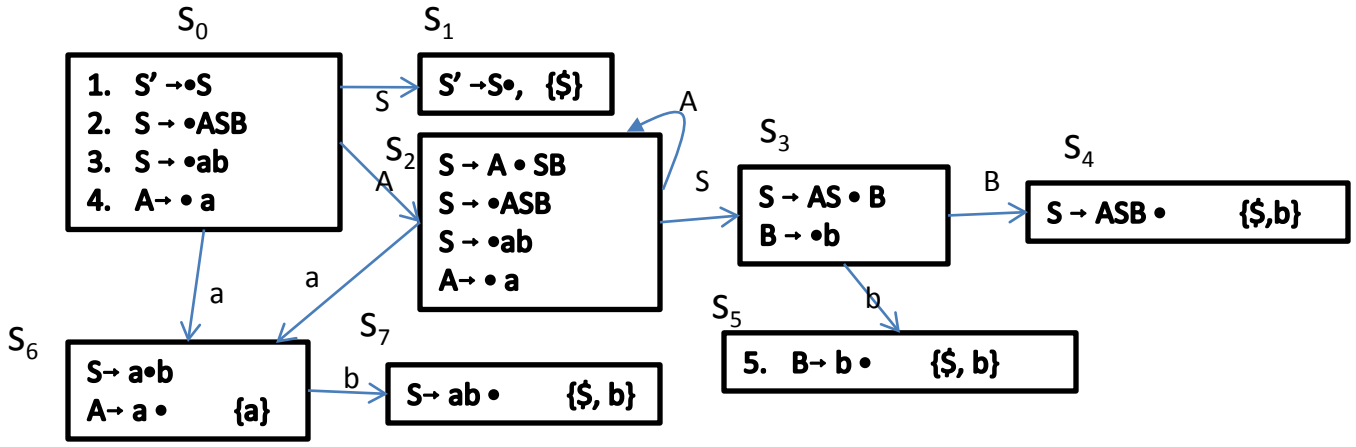


Follow(S') = {$}
Follow(S) = {$}
Follow(A) = {a, b}
Follow(B) = {a, b}

It is NOT SLR.
Reduce-reduce conflict at state S_{10}.

	Action				GOTO		
	a	b	c	$	S	A	B
S_0			s10		1	2	6
S_1				!			
S_2	s3						
S_3			s11			4	
S_4		s5					
S_5				r2			
S_6		s7					
S_7			s12				8
S_8	s9						
S_9				r3			
S_{10}	r4 or r5? conflict	r4 or r5? conflict					
S_{11}	r4	r4					
S_{12}	r5	r5					

2. (c)



	Action			GOTO		
	a	b	$	S	A	B
S_0	s6			1	2	
S_1			!			
S_2	s6			3	2	
S_3		s5				4
S_4		r2	r2			
S_5		r5	r5			
S_6	r4	s7				
S_7		r3	r3			

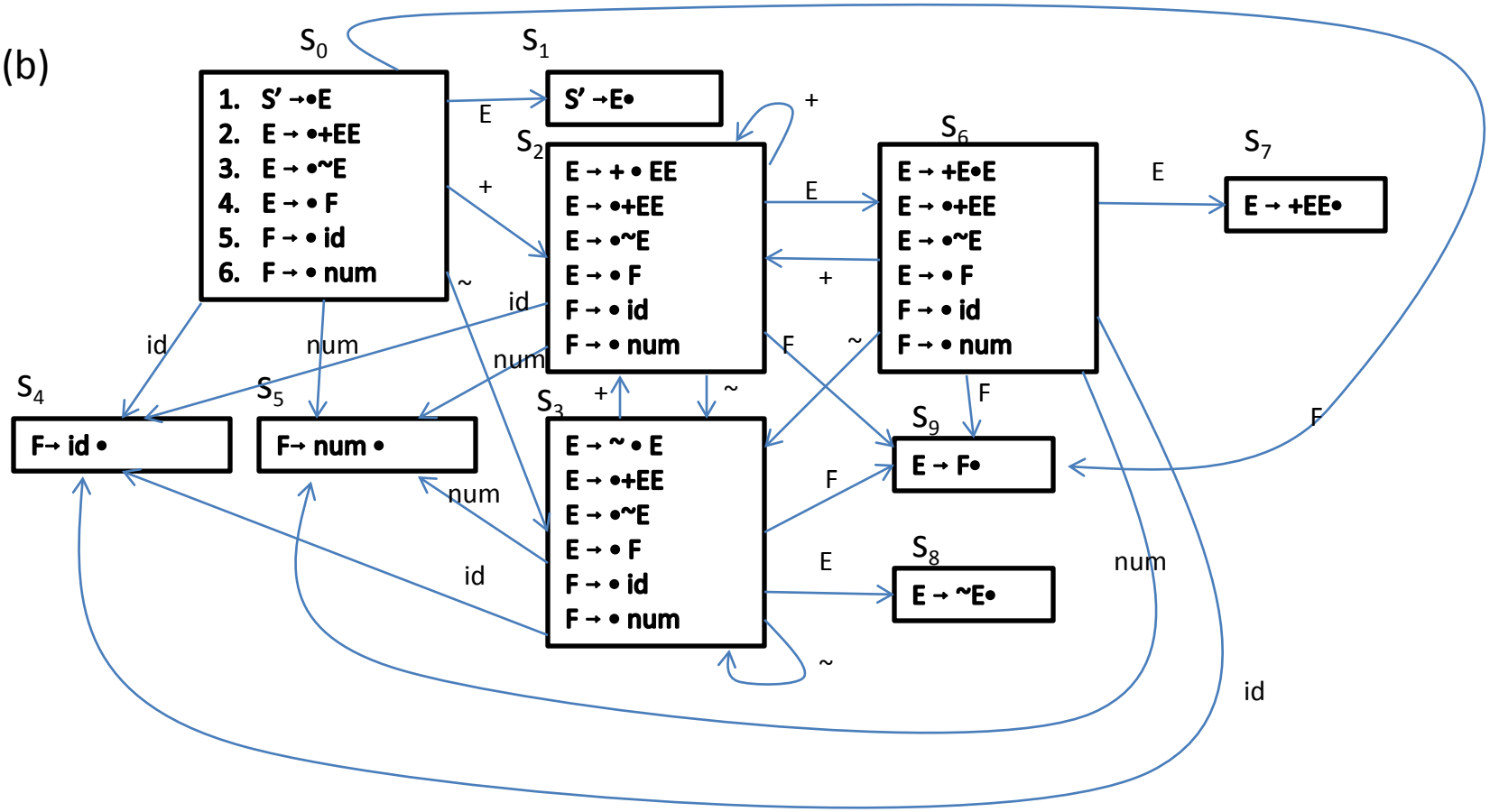
Follow(S') = {$}
Follow(S) = {$, b}
Follow(A) = {a}
Follow(B) = {$, b}

It is SLR. No conflicts.

3. (a)

S’
⇒ E
⇒ + E E
⇒ + E F
⇒ + E id_{cy}
⇒ + + E E id_{cy}
⇒ + + E F id_{cy}
⇒ + + E num_{12} id_{cy}
⇒ + + ~ F num_{12} id_{cy}
⇒ + + ~ num_5 num_{12} id_{cy}

3. (b)



3. (c)

	First	Follow
S'	{+, ~, id, num}	{$}
E	{+, ~, id, num}	{+, ~, id, num, $}
F	{id, num}	{+, ~, id, num, $}

3. (d)

	Action					GOTO	
	+	~	id	num	$	E	F
S_0	s2	s3	s4	s5		1	9
S_1					!		
S_2	s2	s3	s4	s5		6	9
S_3	s2	s3	s4	s5		8	9
S_4	r5	r5	r5	r5	r5		
S_5	r6	r6	r6	r6	r6		
S_6	s2	s3	s4	s5		7	9
S_7	r2	r2	r2	r2	r2		
S_8	r3	r3	r3	r3	r3		
S_9	r4	r4	r4	r4	r4		

3. (e)

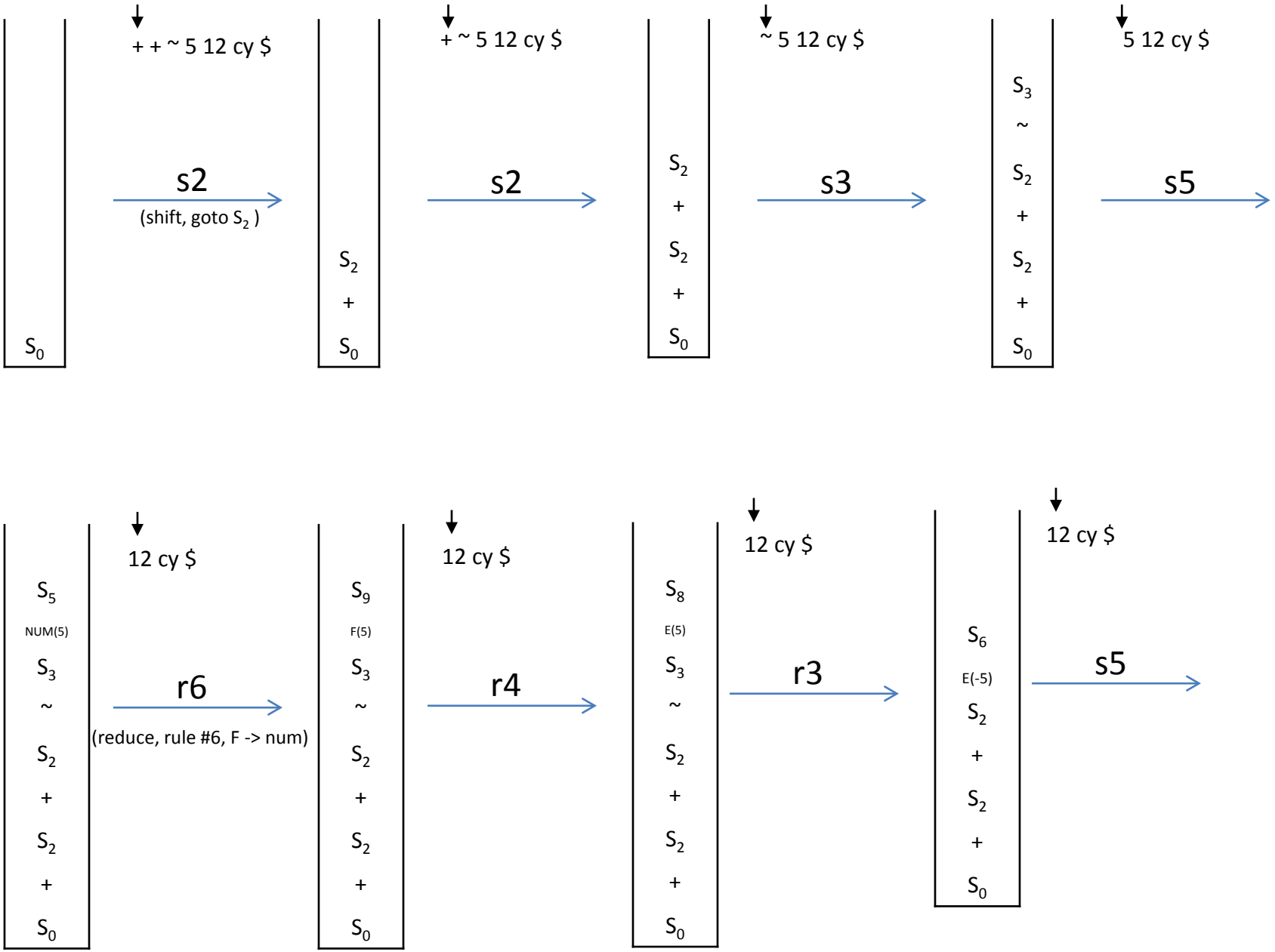
Flex

```
%%
[0-9]+      { yylval = atoi(yytext);
              return NUM;}

[a-z]+      { int i;
              yylval = 0;
              for (i=0; i< yyleng; i++)
                 yylval = yylval*26 + yytext[i]-'a';
              return ID;
            }
%%
```

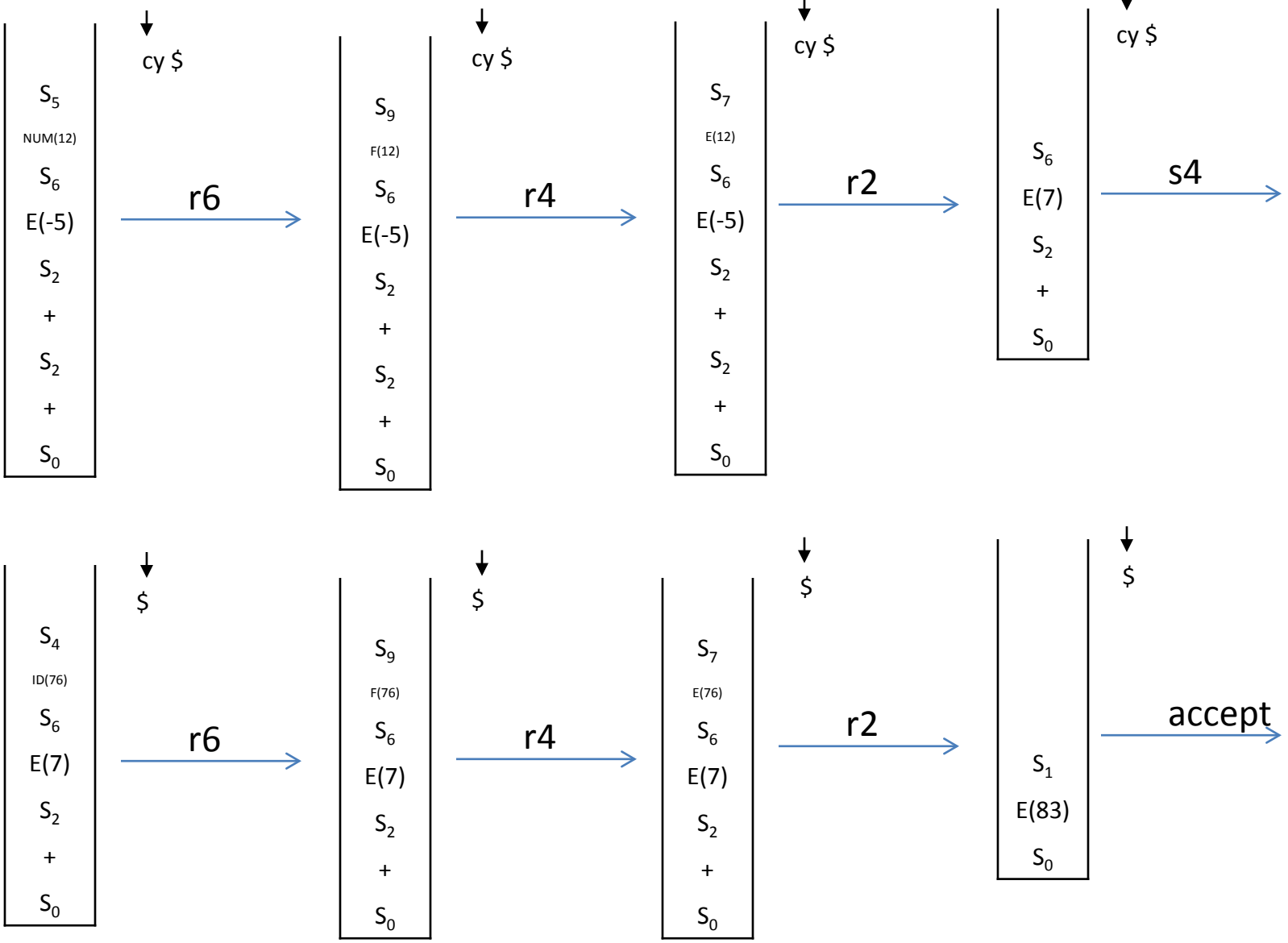
Bison

```
%%
S : E  { $$ = $1; printf("%d\n", $1}} ;

E : '+' E E { $$ = $2+$3;}
  | '~' E   {$$ = -$2;}
  | F { $$ = $1; }
  ;

F : ID {$$ = $1;}
  | NUM {$$ = $1;}
  ;

%%
```

3(f)


+ + ~ 5 12 cy $
s2
(shift, goto S_2)
S_0
+ ~ 5 12 cy $
s2
S_2 + S_0
~ 5 12 cy $
s3
S_2 + S_2 + S_0
5 12 cy $
s5
S_3 ~ S_2 + S_2 + S_0
12 cy $
r6
(reduce, rule #6, F -> num)
S_5 NUM(5) S_3 ~ S_2 + S_2 + S_0
12 cy $
r4
S_9 F(5) S_3 ~ S_2 + S_2 + S_0
12 cy $
r3
S_8 E(5) S_3 ~ S_2 + S_2 + S_0
12 cy $
s5
S_6 E(-5) S_2 + S_2 + S_0

3(f)cont.


cy $
S5
NUM(12)
S6
E(-5)
S2
+
S2
+
S0
r6
cy $
S9
F(12)
S6
E(-5)
S2
+
S2
+
S0
r4
cy $
S7
E(12)
S6
E(-5)
S2
+
S2
+
S0
r2
cy $
S6
E(7)
S2
+
S0
s4
$
S4
ID(76)
S6
E(7)
S2
+
S0
r6
$
S9
F(76)
S6
E(7)
S2
+
S0
r4
$
S7
E(76)
S6
E(7)
S2
+
S0
r2
$
S1
E(83)
S0
accept

3. (g) The parsing process is the reverse of the rightmost derivation.