2.1 Write regular expressions for each of the following.

a. Strings over the alphabet {a, b, c} where the first a precedes the first b.

c*a(a|c)*b[a-c]*

b. Strings over the alphabet {a, b, c} with an even number of a's.

((b|c)*a(b|c)*a)* (b|c)*, or (b|c)* (a(b|c)*a(b|c)*)*

c. Binary numbers that are multiples of four.

(1|0)*00, or (1(1|0)*00)|0

d. Binary numbers that are greater than 101001.

10101(0|1) | 1011(0|1)(0|1) | 11(0|1)(0|1)(0|1)(0|1) |

(0|1)*1 (0|1)*(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)

e. Strings over the alphabet {a, b, c} that don't contain the contiguous substring baa.

(a|c)*(b|bc(a|c)*|ba|bac(a|c)*)*

f. The language of nonnegative integer constants in C, where numbers beginning with 0 are octal constants and other numbers are decimal constants.

(00|0[1-7][0-7]*)|(0|[1-9][0-9]*)

g. Binary numbers n such that there exists an integer solution of $a^n+b^n = c^n$.

1|10

2.2 For each of the following, explain why you're not surprised that there is no regular expression defining it.

a. Strings of *a*'s and *b*'s where there are more *a*'s than *b*'s.

all operations of RE cannot count on letters.

b. Strings of *a*'s and *b*'s that are palindromes (the same forward as backward).

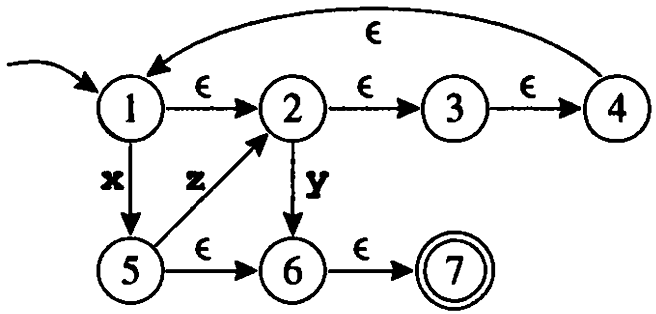
No operation can repeat the string inversely.

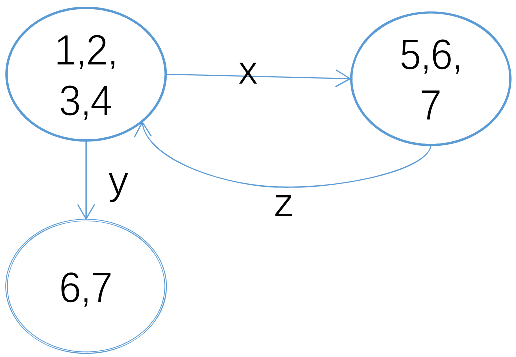
c. Syntactically correct C programs.

balanced parenthesis cannot be expressed by RE.

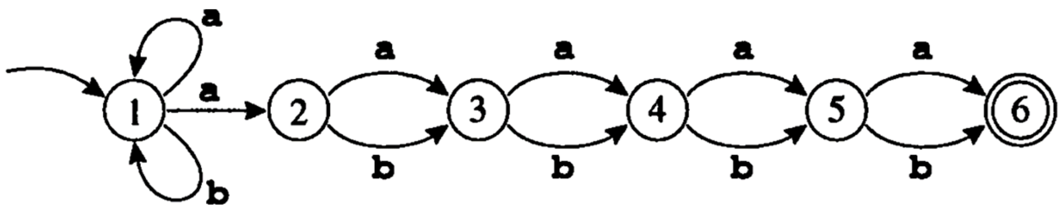
2.5 Convert these NFAs to deterministic finite automata.

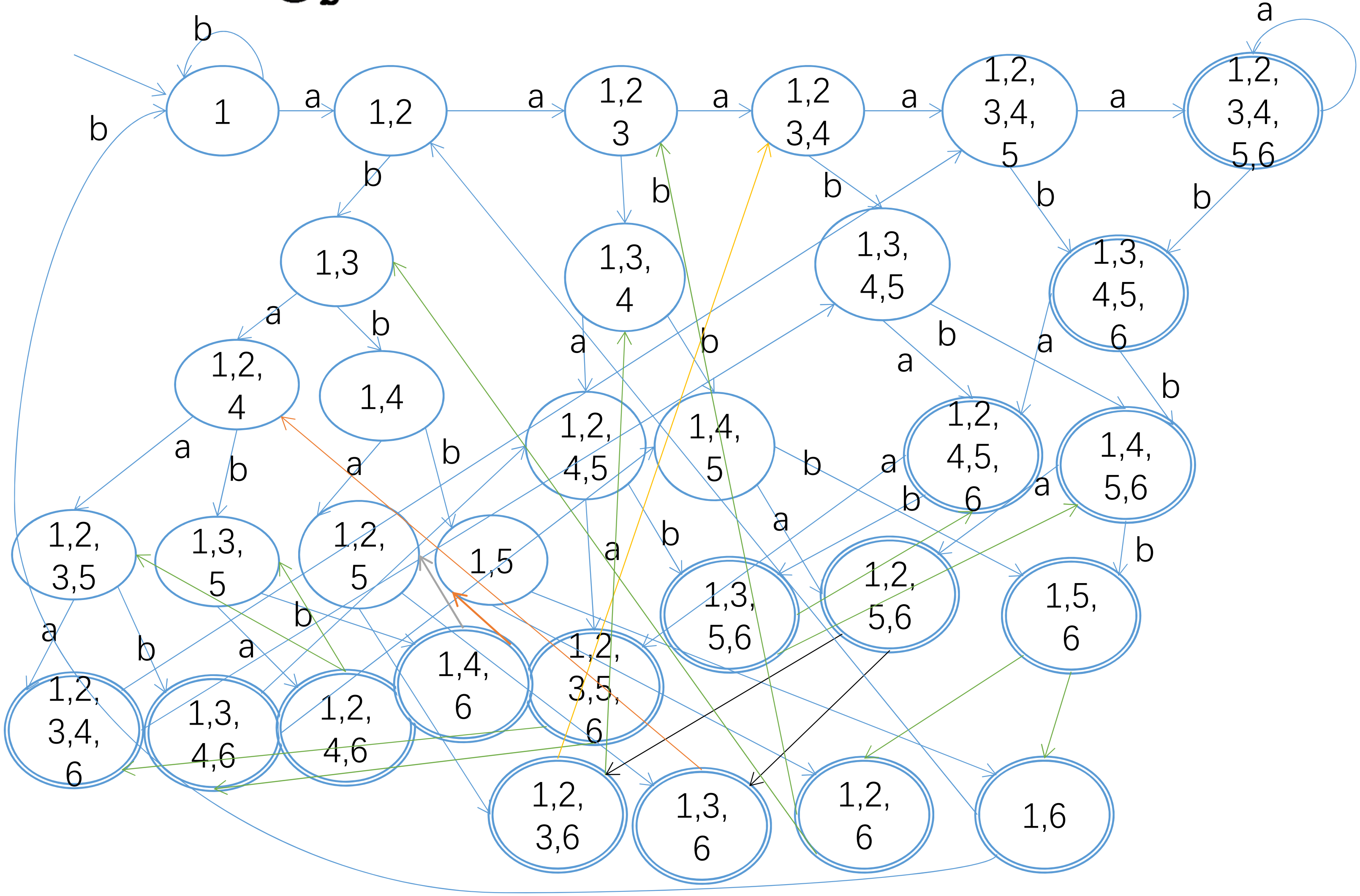
a.



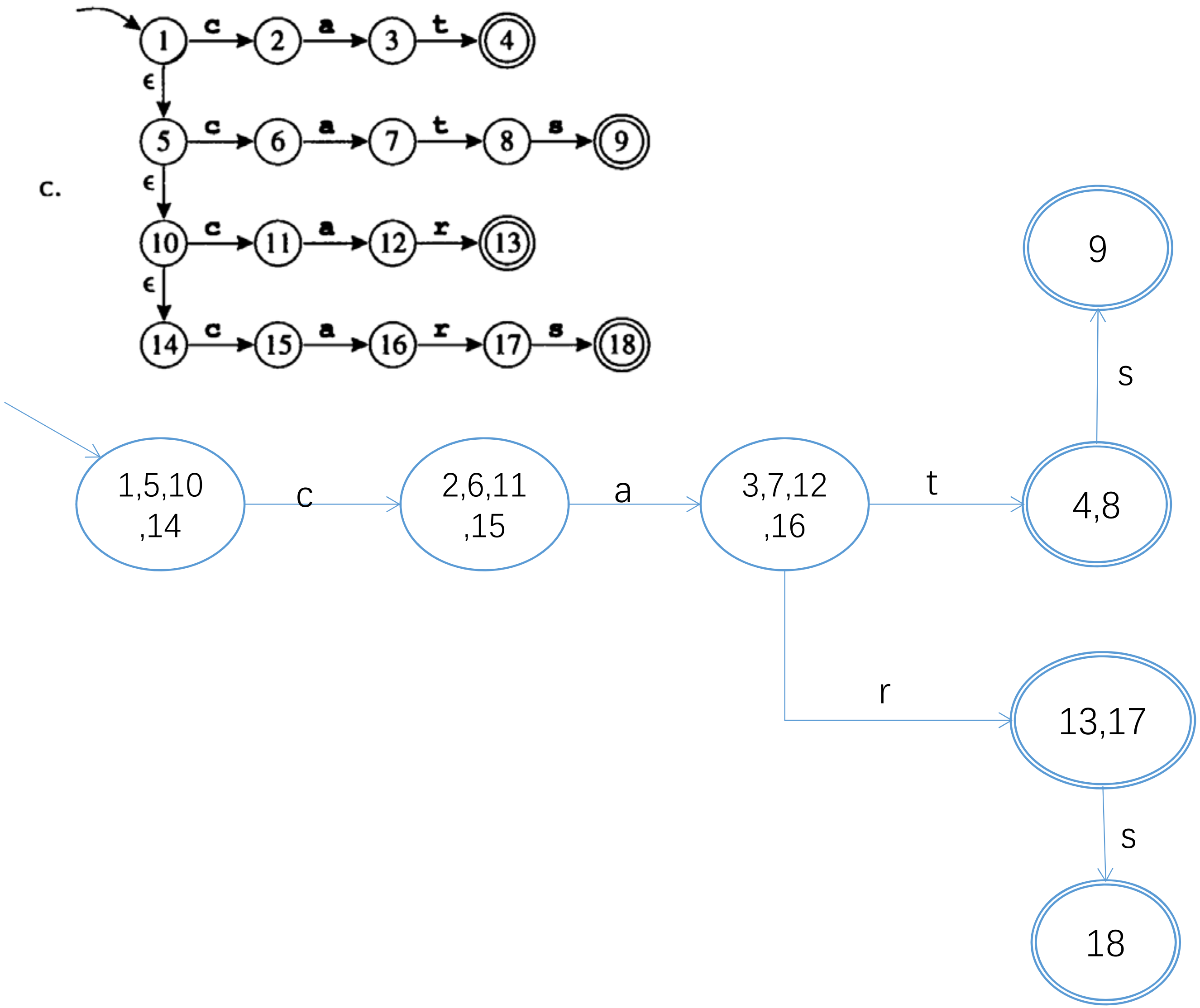


b.

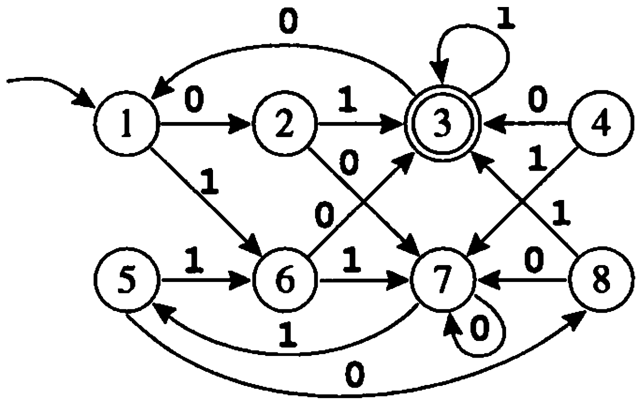

1
2
3
4
5
6
a
b


1
1,2
1,2 3
1,2 3,4
1,2, 3,4, 5
1,2, 3,4, 5,6
1,3
1,3, 4
1,3, 4,5
1,3, 4,5, 6
1,2, 4
1,4
1,2, 4,5
1,4, 5
1,2, 4,5, 6
1,4, 5,6
1,2, 3,5
1,3, 5
1,2, 5
1,5
1,3, 5,6
1,2, 5,6
1,5, 6
1,2, 3,4, 6
1,3, 4,6
1,2, 4,6
1,4, 6
1,2, 3,5, 6
1,2, 3,6
1,3, 6
1,2, 6
1,6
a
b

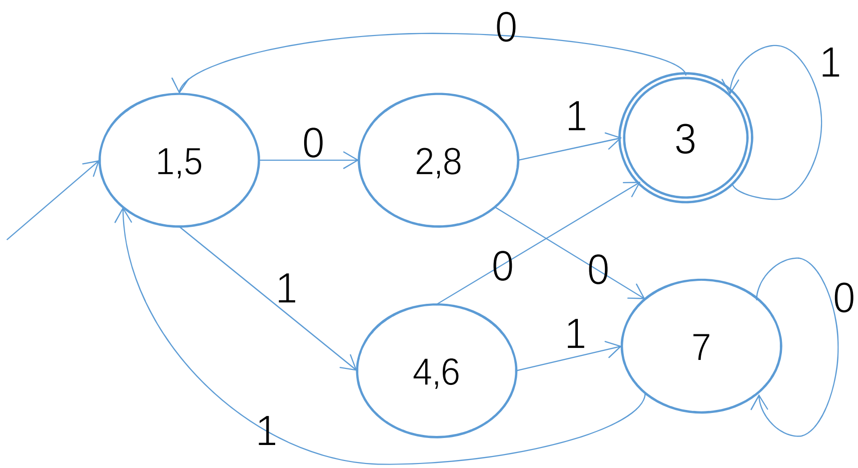
c.


1 c 2 a 3 t 4
ϵ
5 c 6 a 7 t 8 s 9
ϵ
10 c 11 a 12 r 13
ϵ
14 c 15 a 16 r 17 s 18
1,5,10 ,14
c
2,6,11 ,15
a
3,7,12 ,16
t
4,8
s
9
r
13,17
s
18

2.6



Equivalent states:
(1, 5)
(2, 8)
(4, 6)



3.6 a.

	nullable	first	follow
S	no	u	$
B	no	w	v, x, y, z
D	yes	x, y	z
E	yes	y	x, z
F	yes	x	z

b.

M[N,T]	u	z	v	w	x	y
S	S→uBDz					
B				B → B v B → w		
D		D → E F			D → E F	D → E F
E		E →			E →	E → y
F		F →			F → x	

c. Give evidence that this grammar is not LL(1).

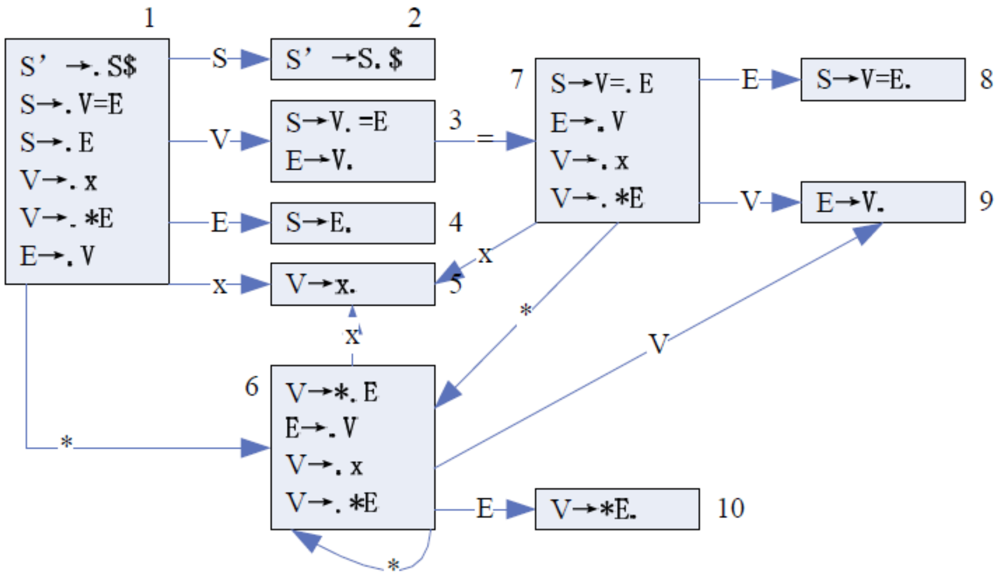
There are two rules in the entry M[B, w] of the parsing table, so the grammar is not LL(1).

d.

After eliminating left recursion, the conflict is removed, so the modified grammar is LL(1).

$$
\begin{array}{l}
B \rightarrow Bv \\
B \rightarrow w
\end{array}
\quad \Rightarrow \quad
\begin{array}{l}
B \rightarrow wB' \\
B' \rightarrow vB' \\
B' \rightarrow
\end{array}
$$

3.9



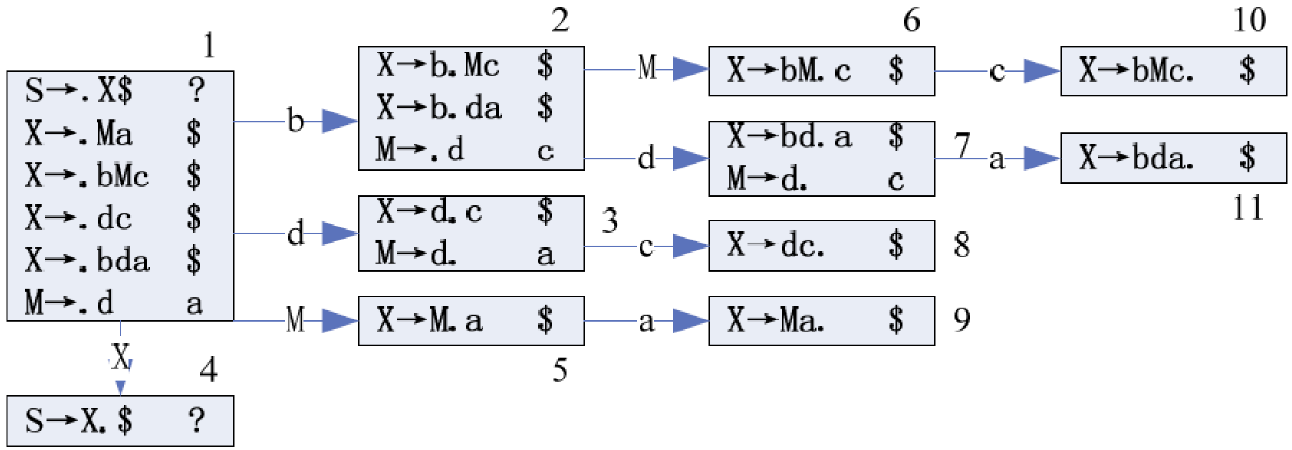
LR(0) DFA of Grammar 3.26

Follow(S) ={$}
Follow(E) ={=, $}
Follow(V) ={=, $}

	=	x	*	$	S	V	E
1		s5	s6		g2	g3	g4
2				accept			
3	**s7, r3**			r3			
4				r2			
5	r4			r4			
6		s5	s6			g9	g10
7		s5	s6			g9	g8
8				r1			
9	r3			r3			
10	r5			r5			

3.13

0. S → X $
1. X → Ma
2. X → bMc
3. X → dc
4. X → bda
5. M → d

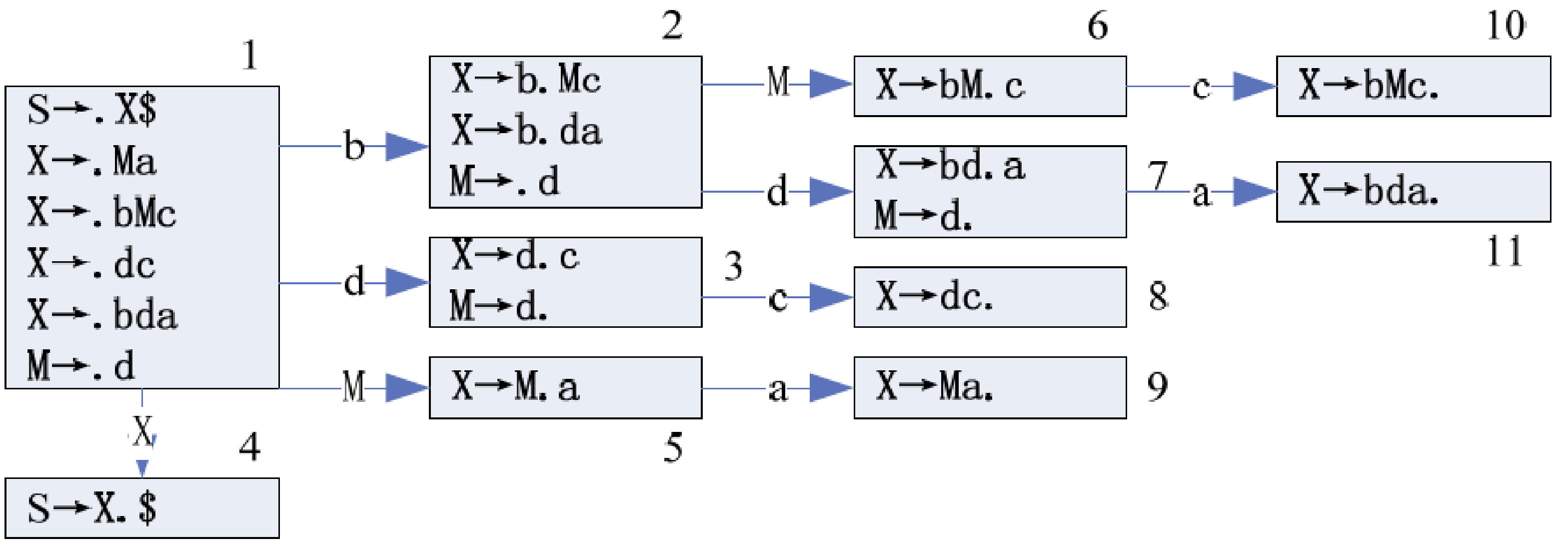


LALR(1) parsing table:

	a	b	c	d	$	X	M
1		s2		s3		g4	g5
2				s7			g6
3	r5		s8				
4					Accept		
5	s9						
6			s10				
7	s11		r5				
8					r3		
9					r1		
10					r2		
11					r4		

There is no conflict in the parsing table, so this grammar is LALR(1).

The LR(0) DFA is:



Follow(S)={}
Follow(X)={$}
Follow(M)={a, c}

	a	b	c	d	$	X	M
1		s2		s3		g4	g5
2				s7			g6
3	r5		**s8, r5**				
4					accept		
5	s9						
6			s10				
7	**s11, r5**		r5				
8					r3		
9					r1		
10					r2		
11					r4		

There are shift-reduce conflict in the parsing table, so this grammar is NOT SLR(1).

3.14

1. S → (X
2. S → E]
3. S → F)
4. X → E)
5. X → F]
6. E → A
7. F → A
8. A →

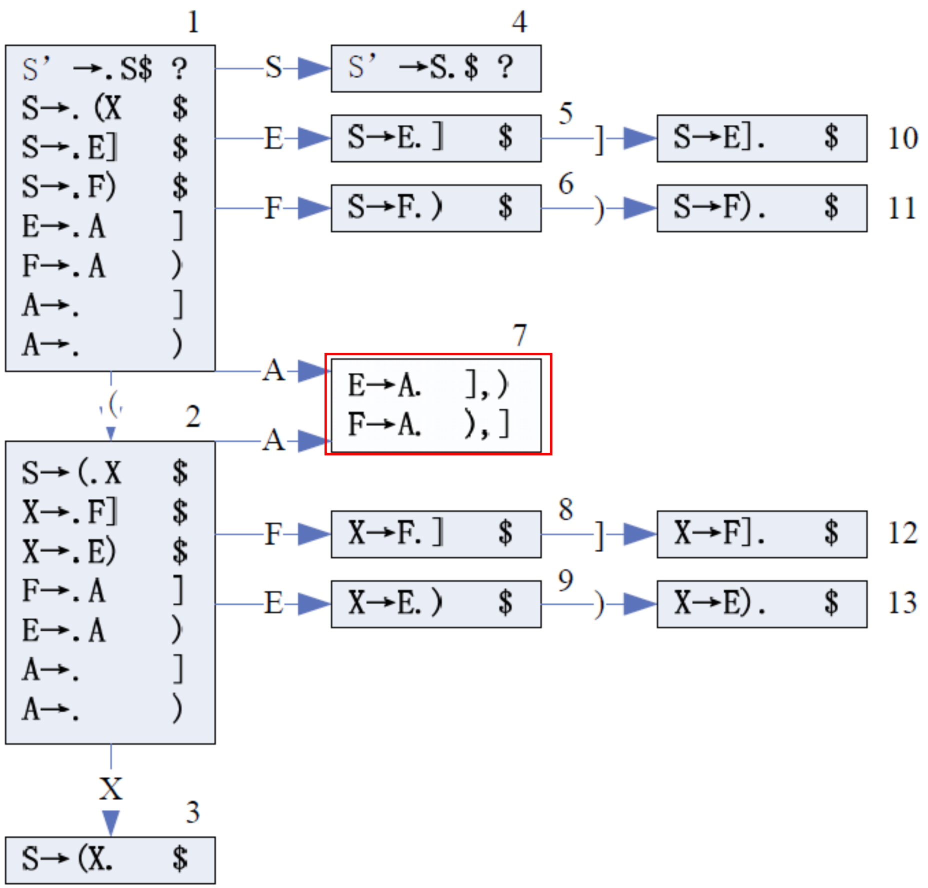
	nullable	First	Follow
S	**no**	(,),]	
X	**no**	),]	
E	**yes**		),]
F	**yes**		),]
A	**yes**		),]

The LL(1) parsing table is:

	(	)	]
S	S→ (X	S→F)	S→E]
X		X→E)	X→F]
E		E→A	E→A
F		F→A	F→A
A		A→	A→

There is no conflict in the parsing table, so this grammar is LL(1).

The LALR(1) DFA is:



There are reduce-reduce conflict in state 7, so this grammar is NOT LALR(1).

6.3

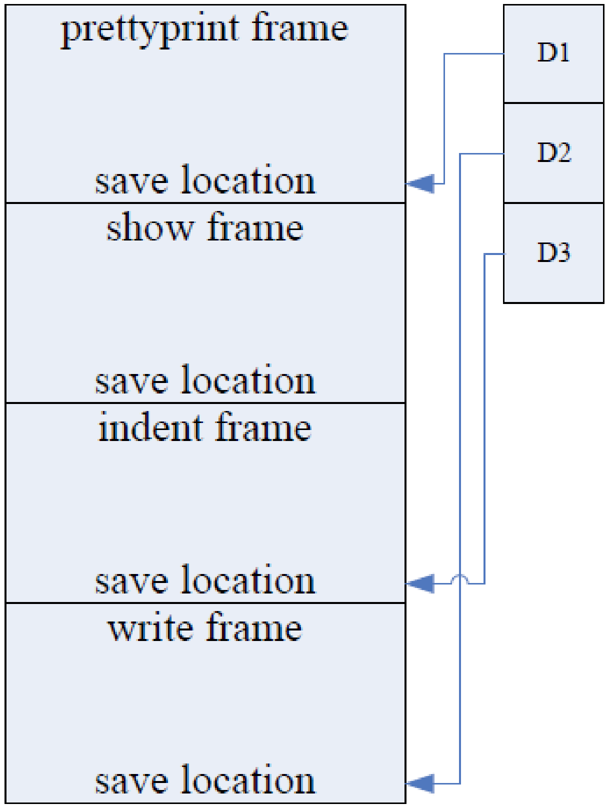
For each a, b, c, d, e should be kept in the memory or register?

variable	**in memory**	**in register**	**reason**
a		√	P132: pass function parameters in registers
b	√		P133: passed by reference, accessed by a procedure
c	√		P133: an array, accessed by a procedure
d		√	P132: intermediate results of expressions, P130: no used after the function g called
e		√	P132: the function result

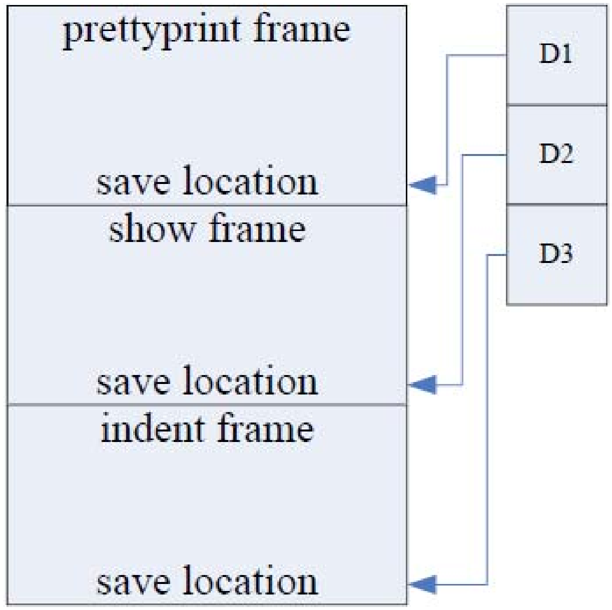
6.7 a. indent uses the variable output from prettyprint's frame. To do so it starts with its own static link:

1. get the frame pointer to the show;
2. then fetches show's static link;
3. get the frame pointer to the prettyprint;
4. then fetches output.

b. indent is at depth 3, when we fetch the variable output in Line14, the display and stack view as below:



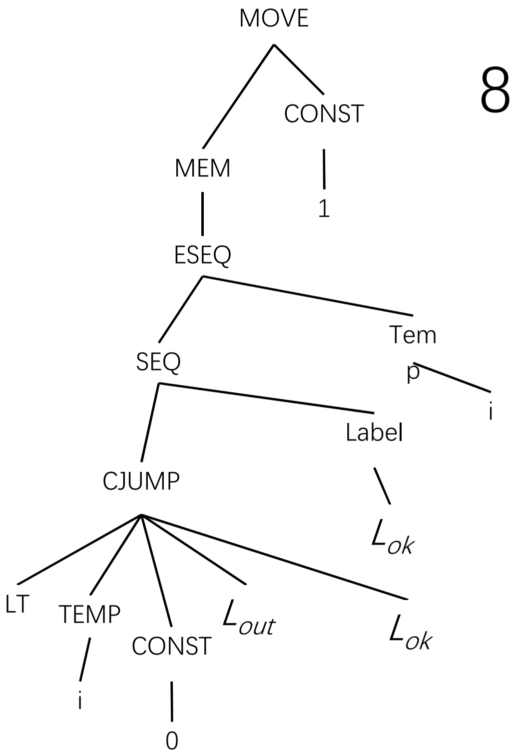
calling ***write*** in ***indent***



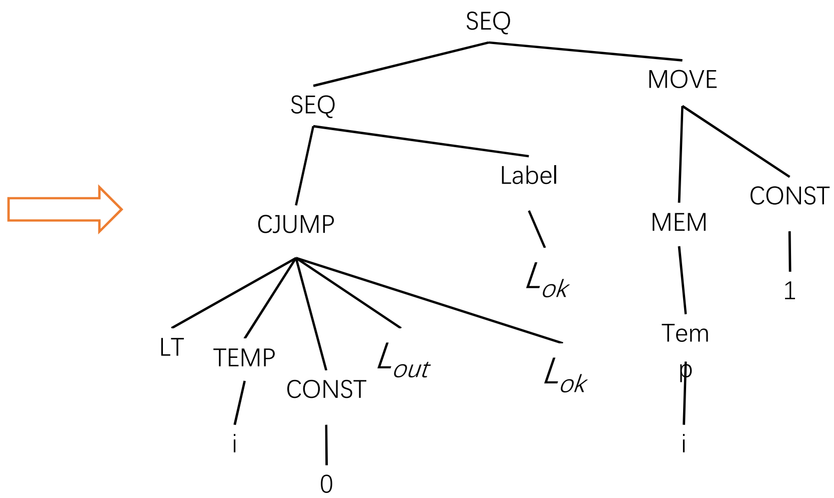
after calling ***write*** in ***indent***, before use the ***output***

When we use the variable ***output***:
1. get the D2 (the frame pointer of ***show***), there is no local variable output;
2. then get D1 (the frame pointer of prettyprint), so fetch the variable output.

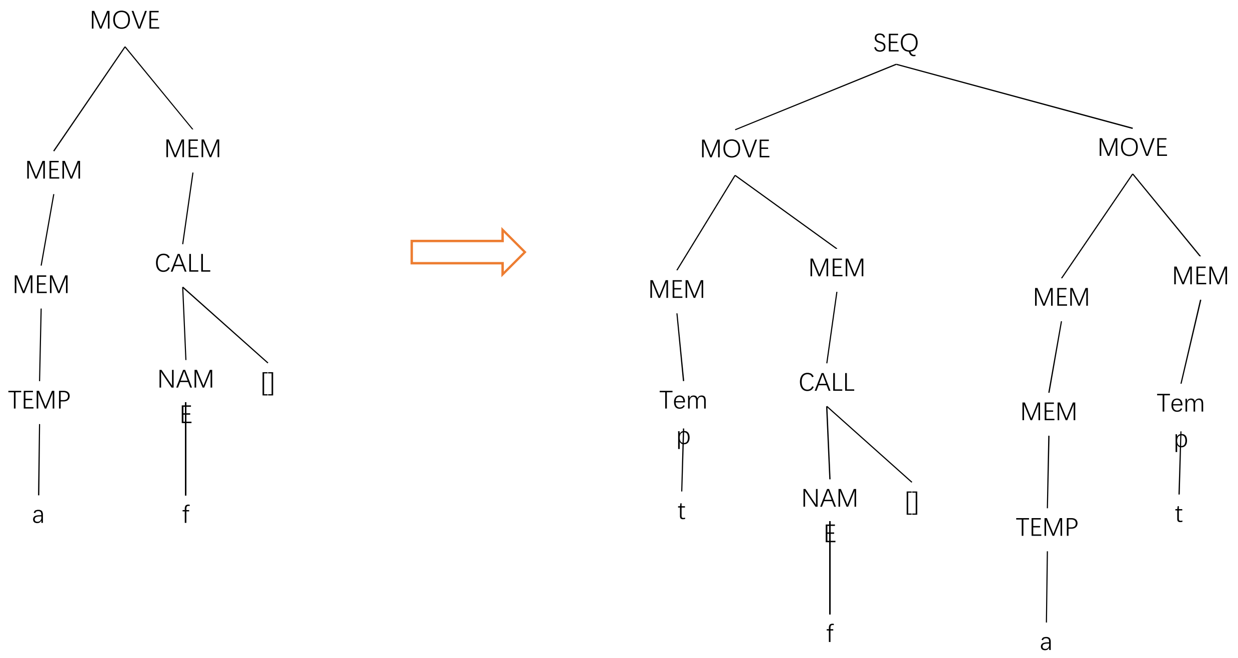
8.2 a. MOVE(MEM(ESEQ(SEQ(CJUMP(LT, TEMP_i , CONST_0, L_{out}, L_{ok}), LABEL_{ok}), TEMP_i)), CONST_1)



8.1 b. MOVE(MEM(ESEQ(s, e1)), e2) ⇒
SEQ(s, MOVE(MEM(e1), e2)

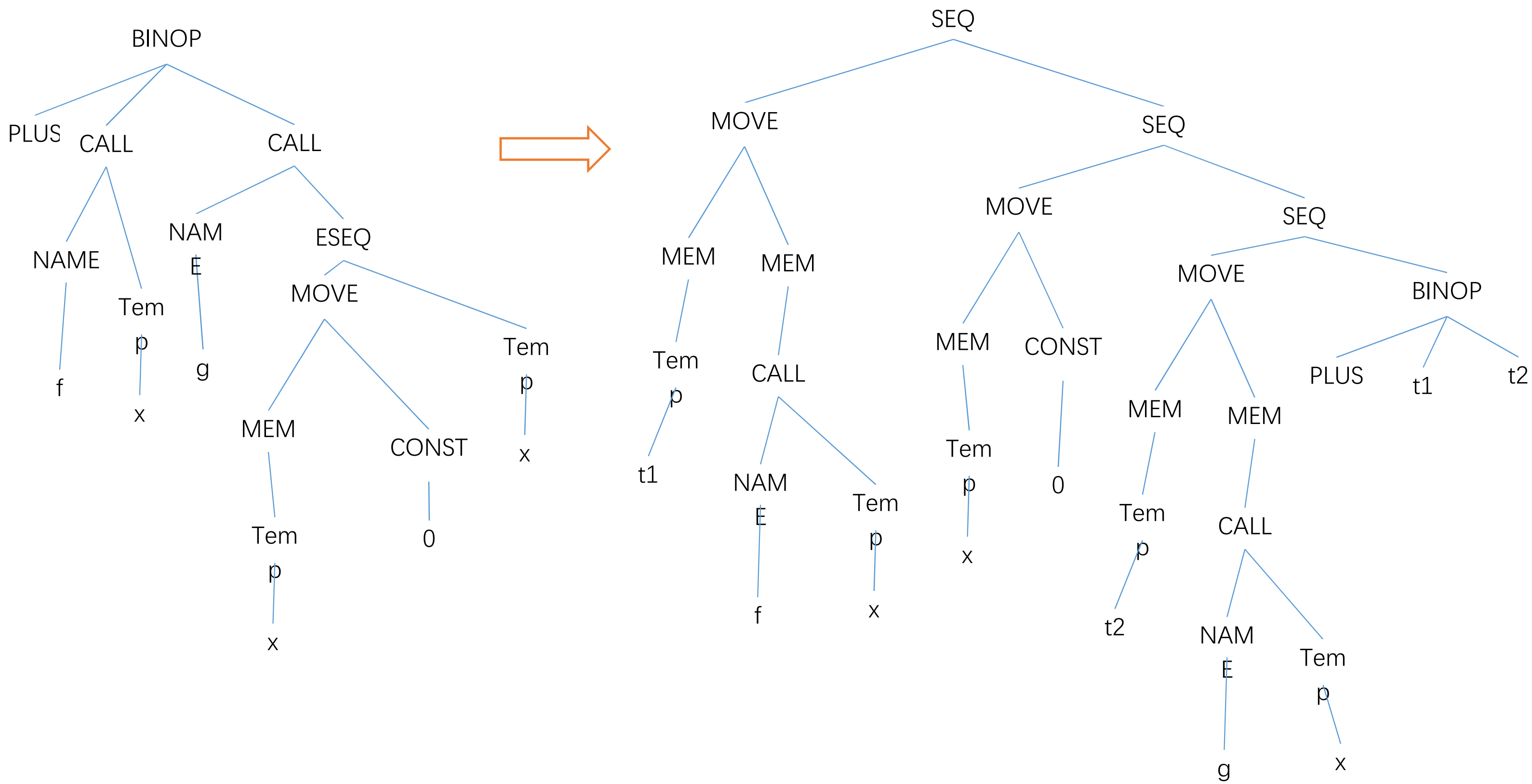


b. MOVE(MEM(MEM(NAME$_a$)), MEM(CALL(TEMP f, [])))

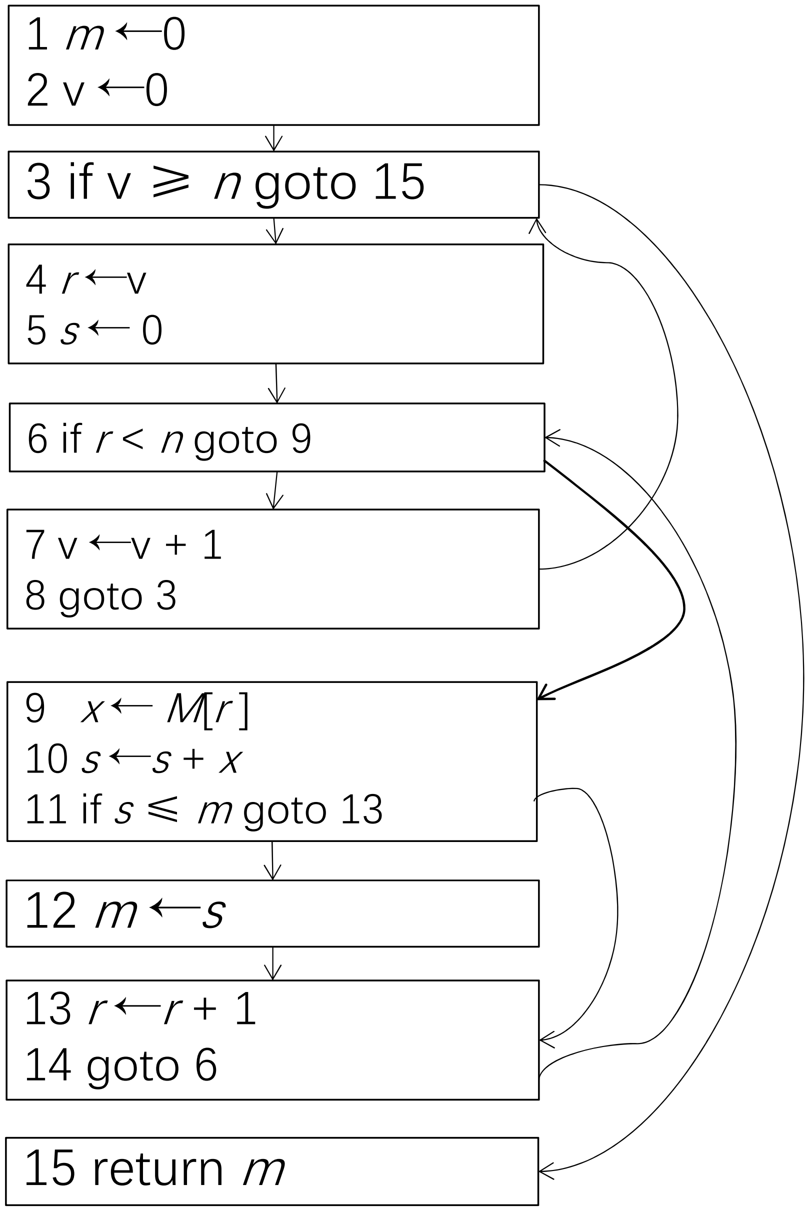


CALL(fun, $args$) $\rightarrow$ ESEQ(MOVE(TEMP t, CALL(fun, $args$)), TEMP t)

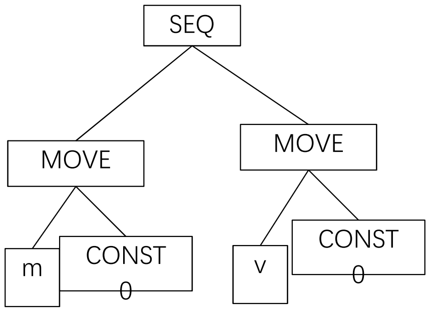
c. BINOP(PLUS, CALL(NAME$_f$, [TEMP$_x$]), CALL(NAME$_g$, [ESEQ(MOVE(TEMP$_x$, CONST$_0$), TEMP$_x$)]))

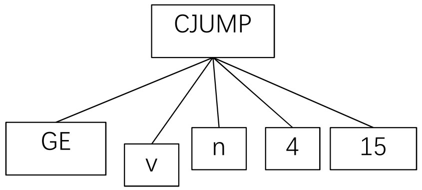


8.6


1 m ← 0
2 v ← 0
3 if v ⩾ n goto 15
4 r ← v
5 s ← 0
6 if r < n goto 9
7 v ← v + 1
8 goto 3
9 x ← M[r]
10 s ← s + x
11 if s ⩽ m goto 13
12 m ← s
13 r ← r + 1
14 goto 6
15 return m

8.7





1 $m \leftarrow 0$
2 v $\leftarrow 0$

3 if v >= n goto 15

4 $r \leftarrow$ v
5 $s \leftarrow 0$

6 if $r < n$ goto 9

7 v $\leftarrow$ v + 1
8 goto 3

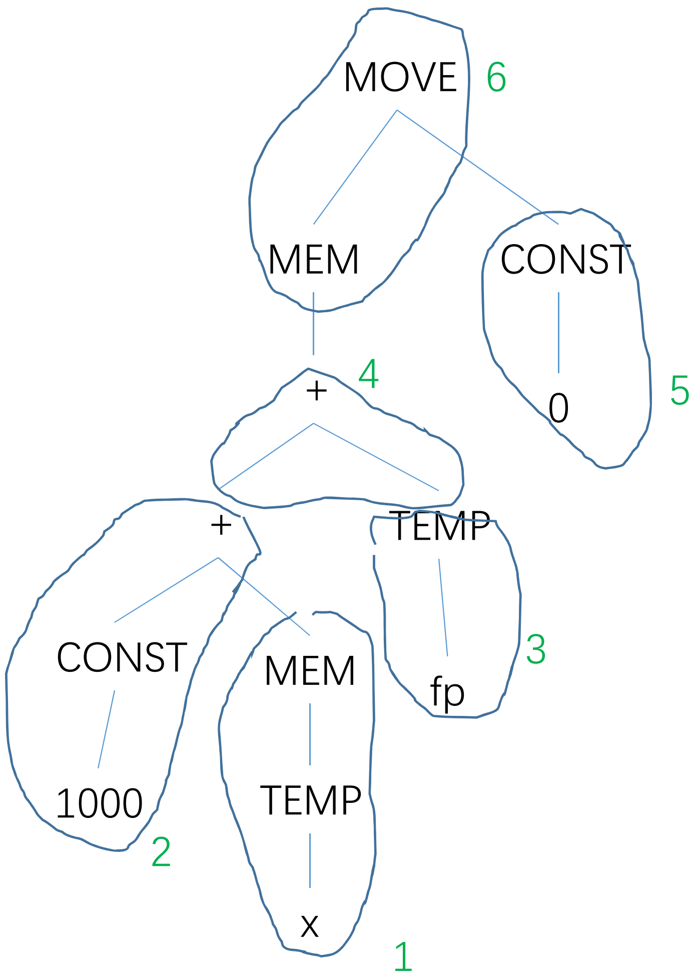
9 $x \leftarrow M[r]$
10 $s \leftarrow s + x$
11 if $s \leqslant m$ goto 13

12 $m \leftarrow s$

13 $r \leftarrow r + 1$
14 goto 6

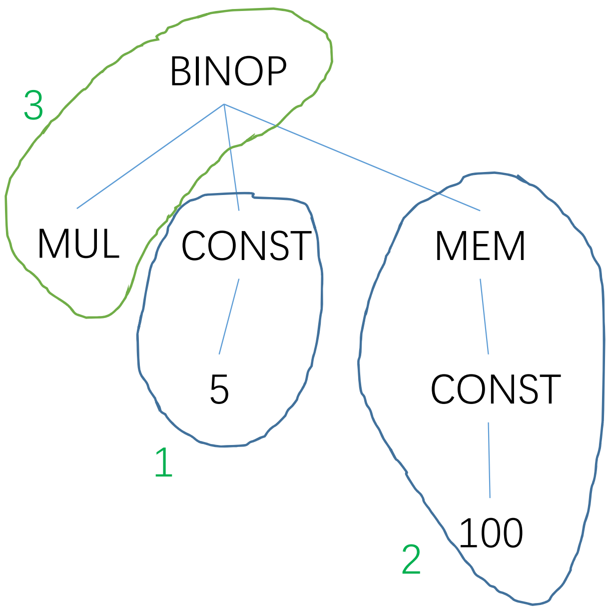
15 return m

9.2 a. MOVE(MEM(+(+(CONST$_{1000}$, MEM(TEMP$_{x}$)), TEMP$_{fp}$)), CONST$_{0}$)



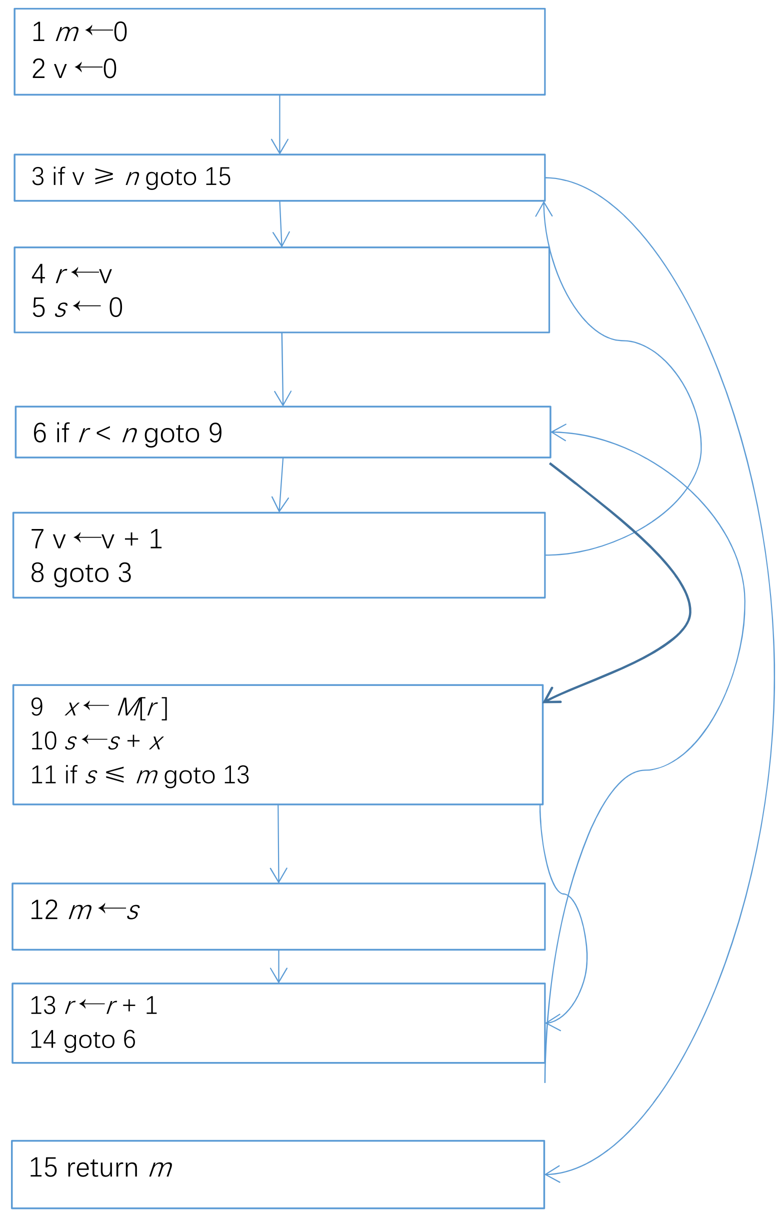
1 LOAD r1←M[r0+0]
2 ADDI r2←r1+1000
3
4 ADD r1←r1+r2
5 ADDI r2←r0+0
6 M[r1+0] ←r2

b. BINOP(MUL, $CONST_5$, MEM($CONST_{100}$))



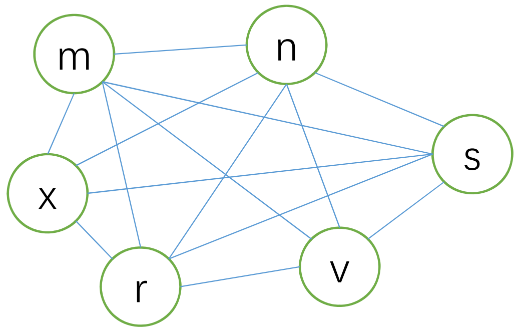
1 ADDI ri←r0+5
2 LOAD rj←M[r0+100]
3 MUL r1←ri+rj

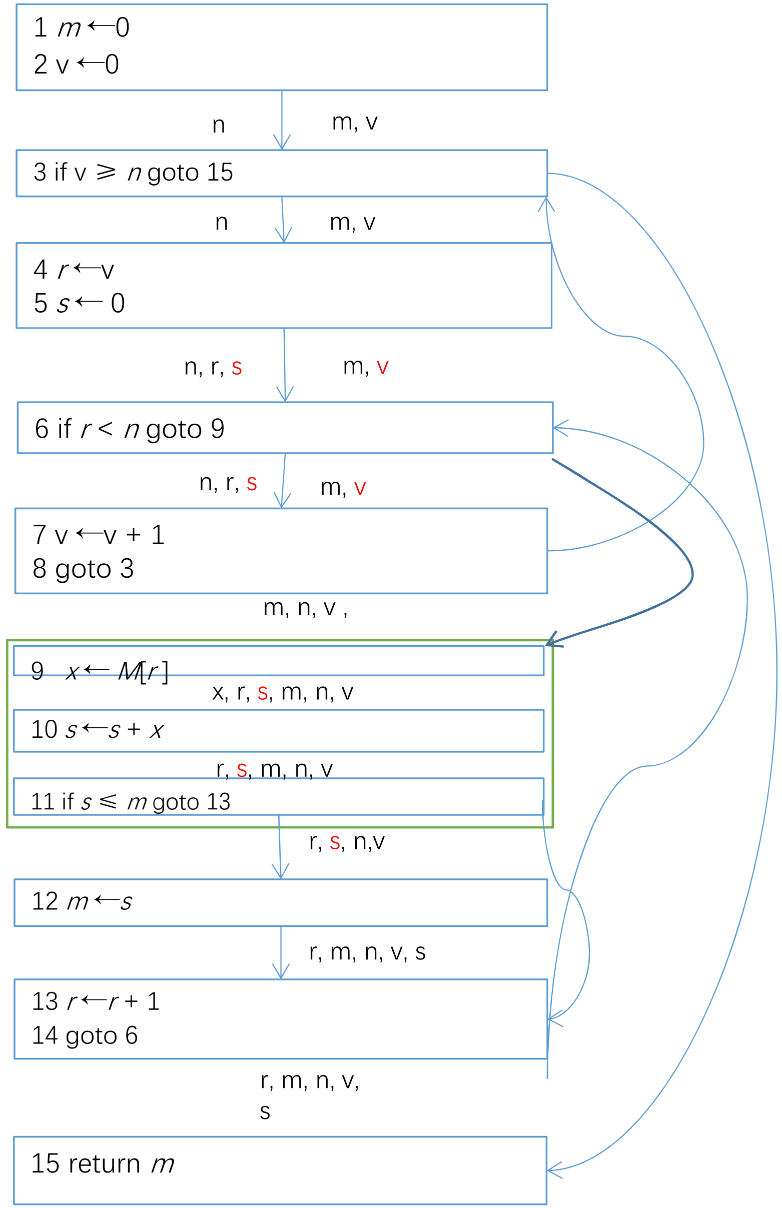
10.1 a. Draw the control-flow graph.



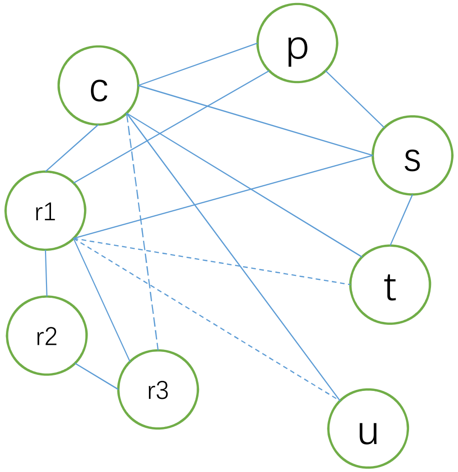
b. Calculate live-in and live-out at each statement.

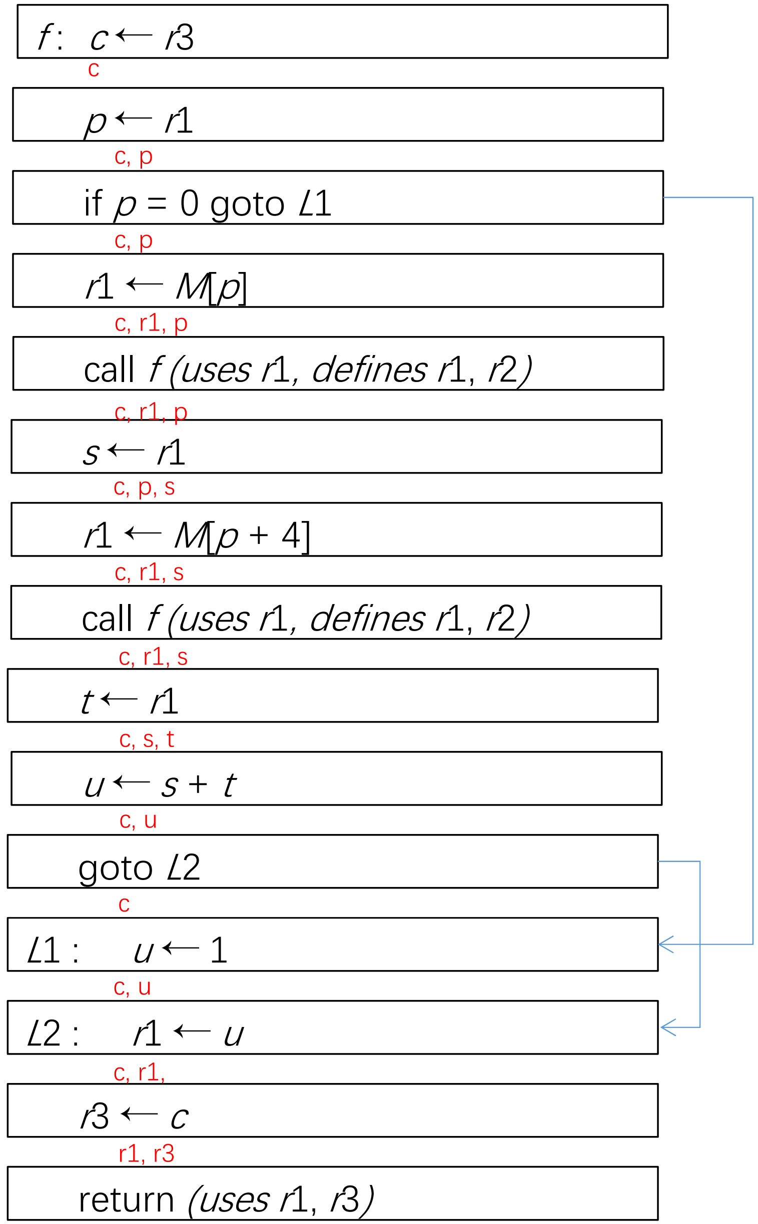
c. Construct the register interference graph.

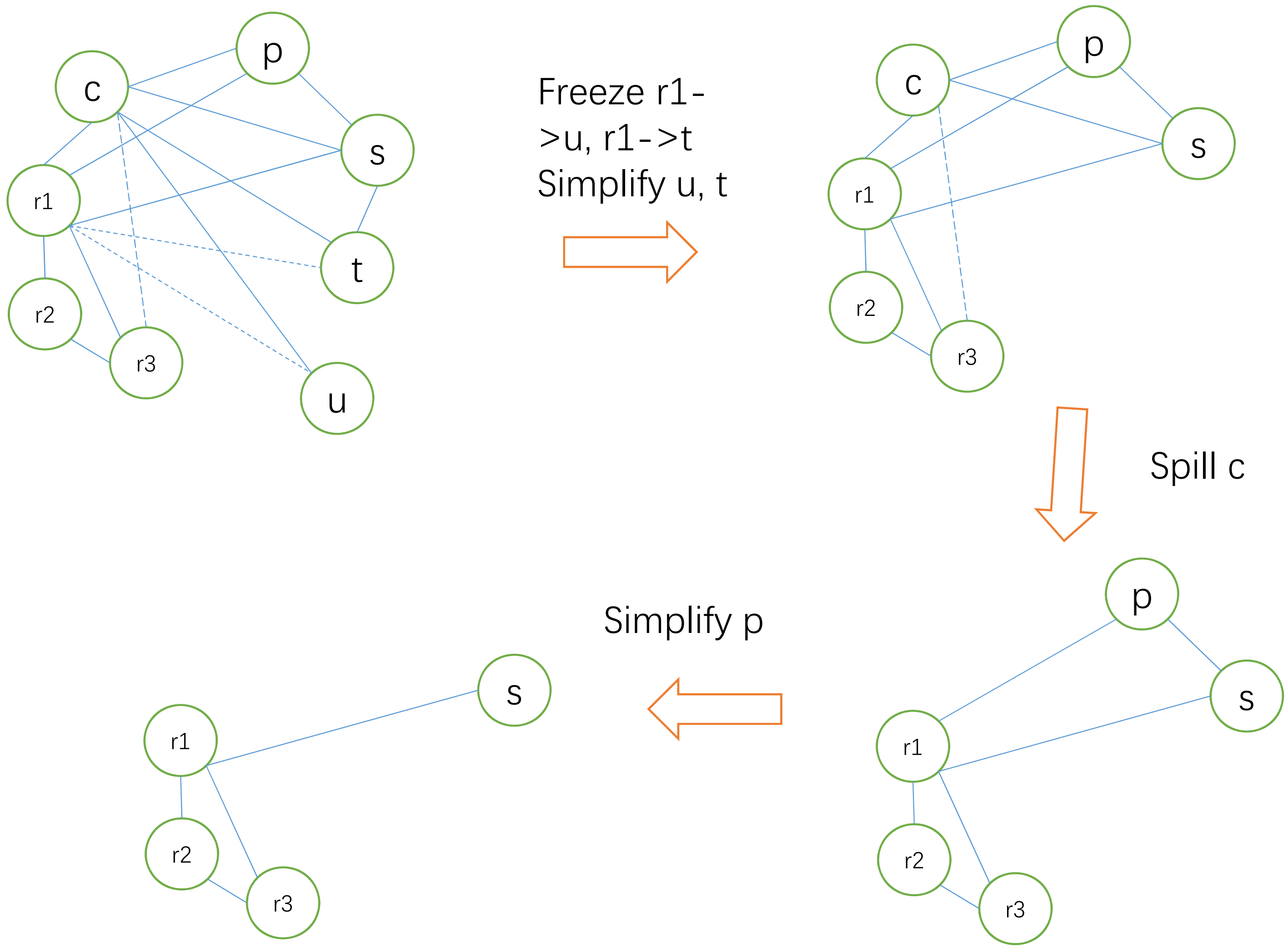




11.1 Solution: The interference graph is:

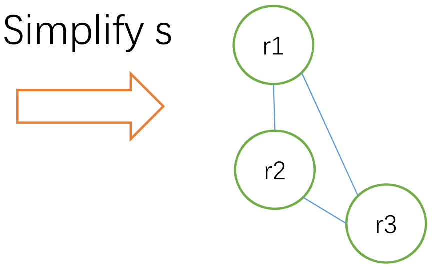




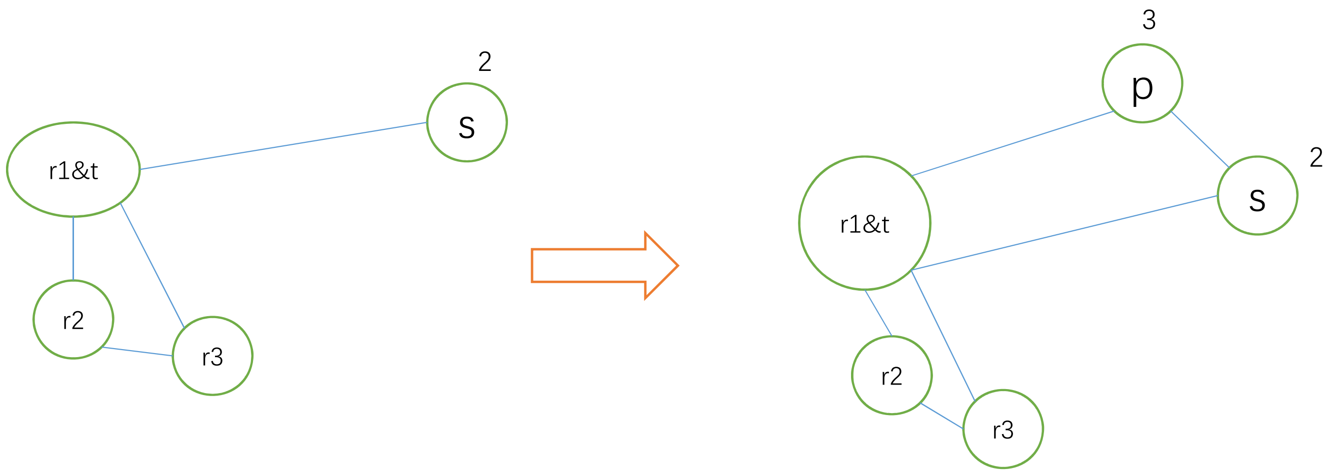


c
p
s
r1
t
r2
r3
u
Freeze r1->u, r1->t
Simplify u, t
c
p
s
r1
r2
r3
Spill c
p
s
r1
r2
r3
Simplify p
s
r1
r2
r3

Simplify s



Coloring:

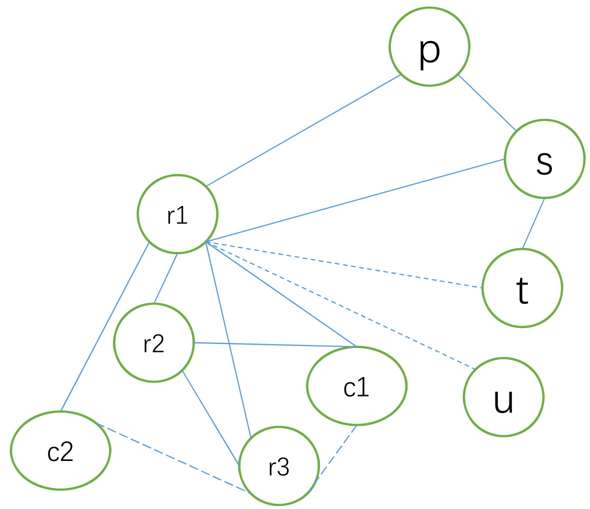


Then c is actually spilled.

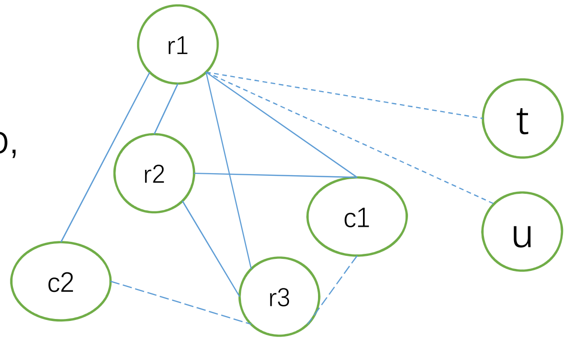
The program is rewritten:

```
f:  c1 ← r3
    M[c_loc] ← c1
    p ← r1
    if p = 0 goto L1
    r1 ← M[p]
    call f (uses r1, defines r1, r2)
    s ← r1
    r1 ← M[p + 4]
    call f (uses r1, defines r1, r2)
    t ← r1
    u ← s + t
    goto L2
L1:     u ← 1
L2:     r1 ← u
    c2 ← M[c_loc]
    r3 ← c2
    return (uses r1, r3)
```

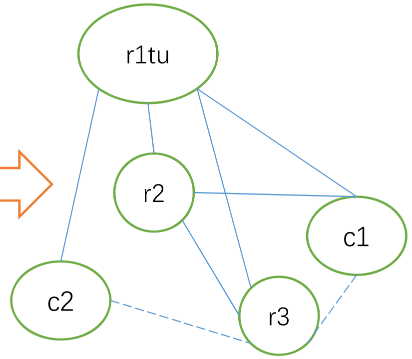
Rebuild the interference graph:



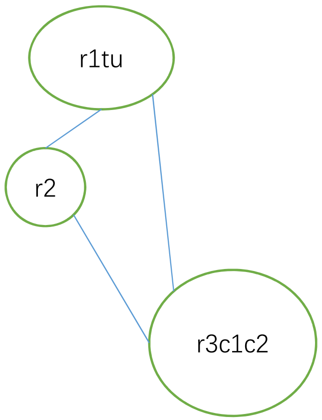
Simplify p, s



Coalesce r1 and t, r1 and u using George criterion



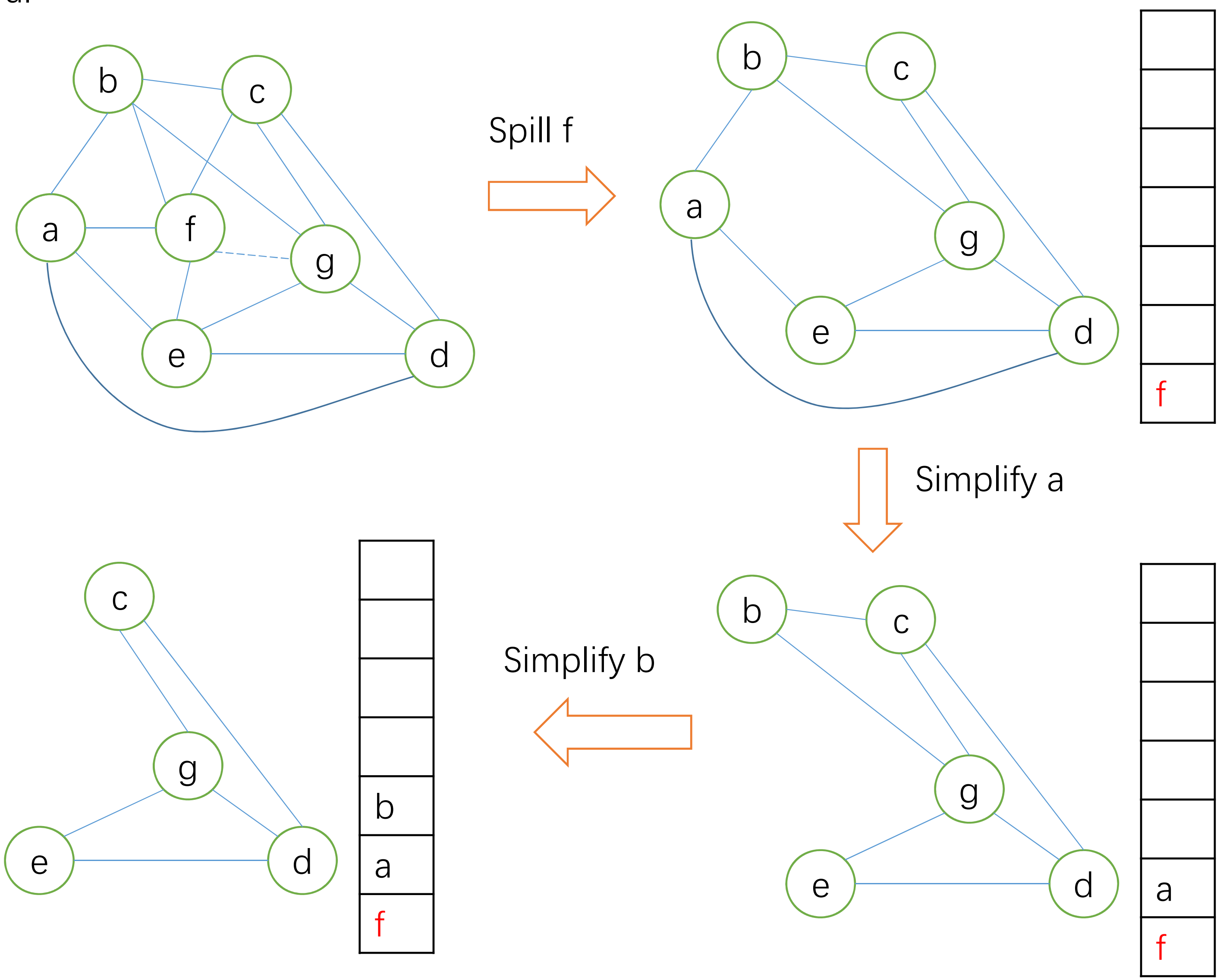
Coalesce r3 and c1, r3 and c2, using Briggs criterion

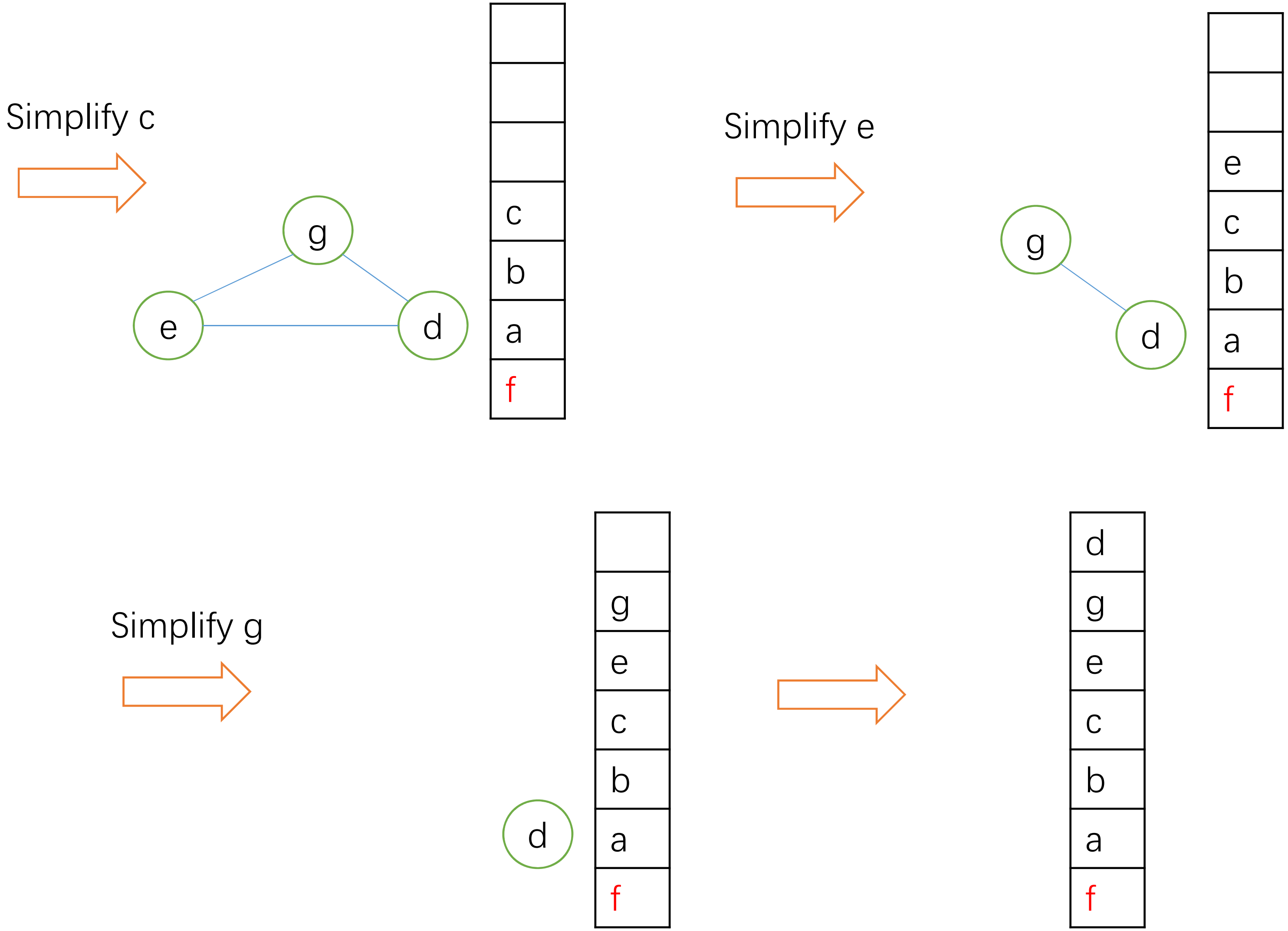


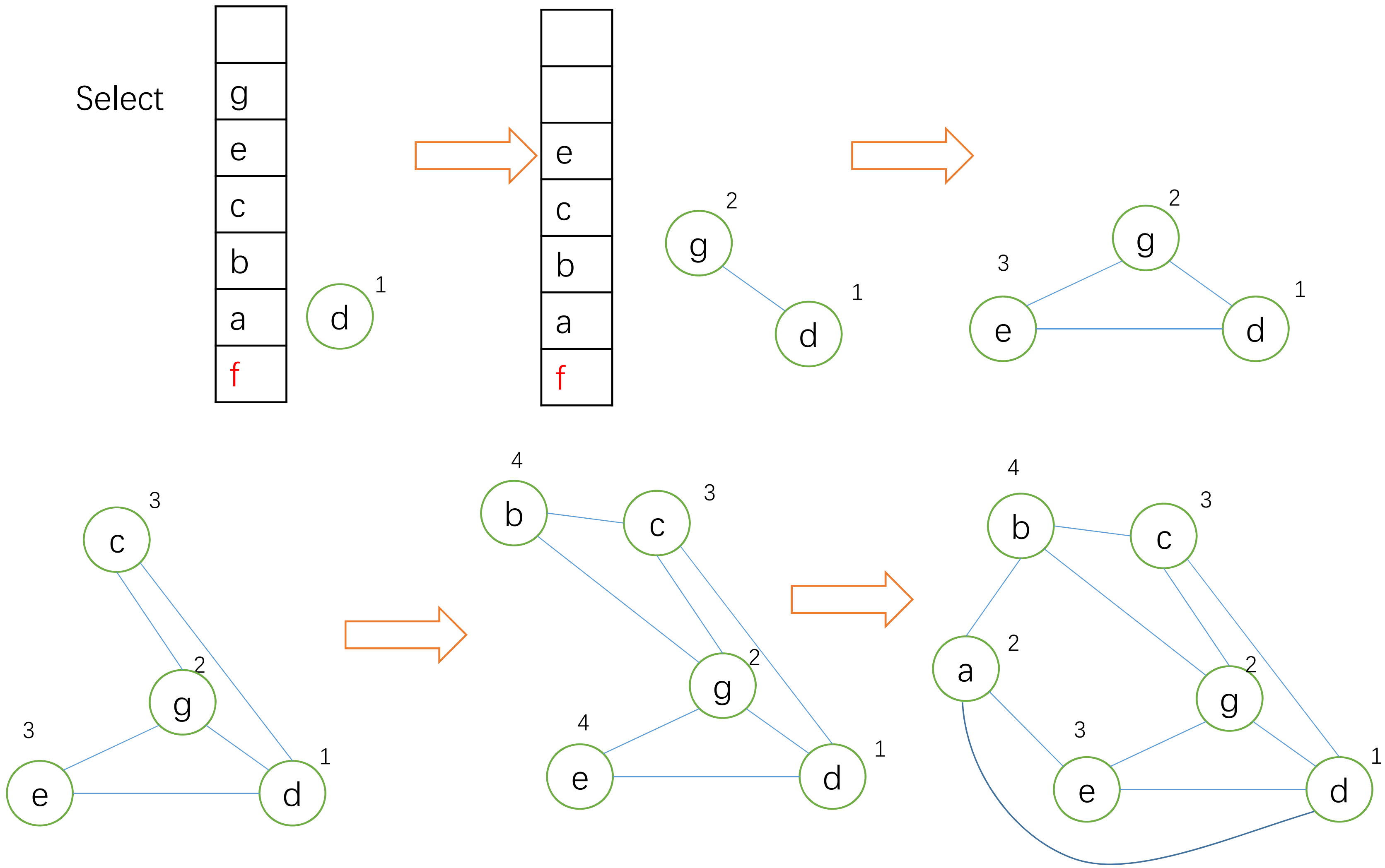
Coloring:

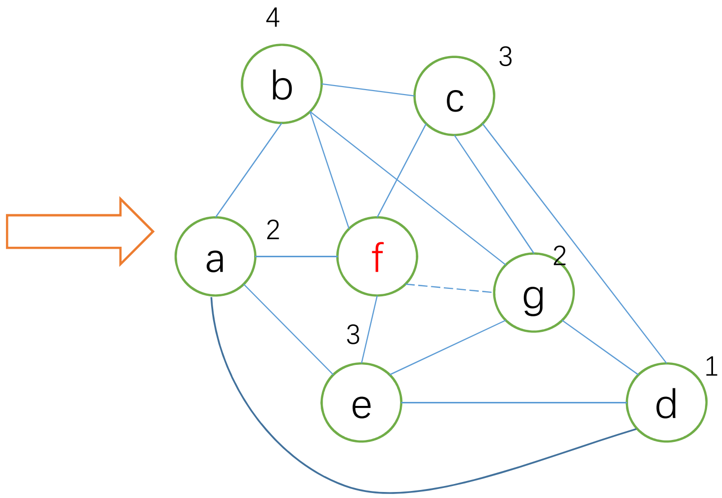
p	3
s	2
u	1
t	1
c1	3
c1	3

11.3 a.



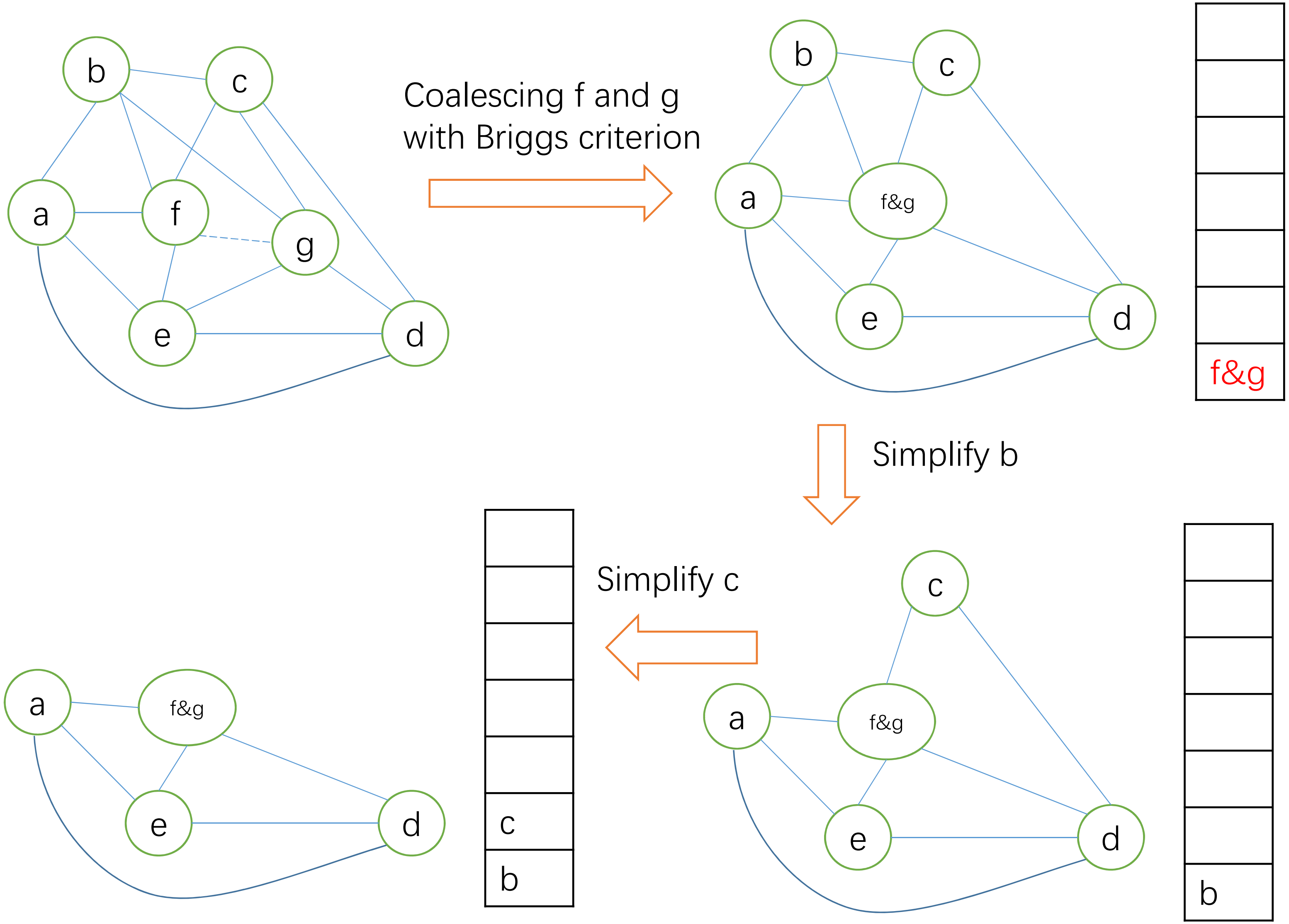

Simplify c
g
e
d
c
b
a
f
Simplify e
e
c
b
a
f
g
d
Simplify g
g
e
c
b
a
f
d
d
g
e
c
b
a
f


Select
g
e
c
b
a
f
d
1
e
c
b
a
f
g
2
d
1
g
2
e
3
d
1
c
3
g
2
e
3
d
1
b
4
c
3
g
2
e
4
d
1
b
4
c
3
a
2
g
2
e
3
d
1



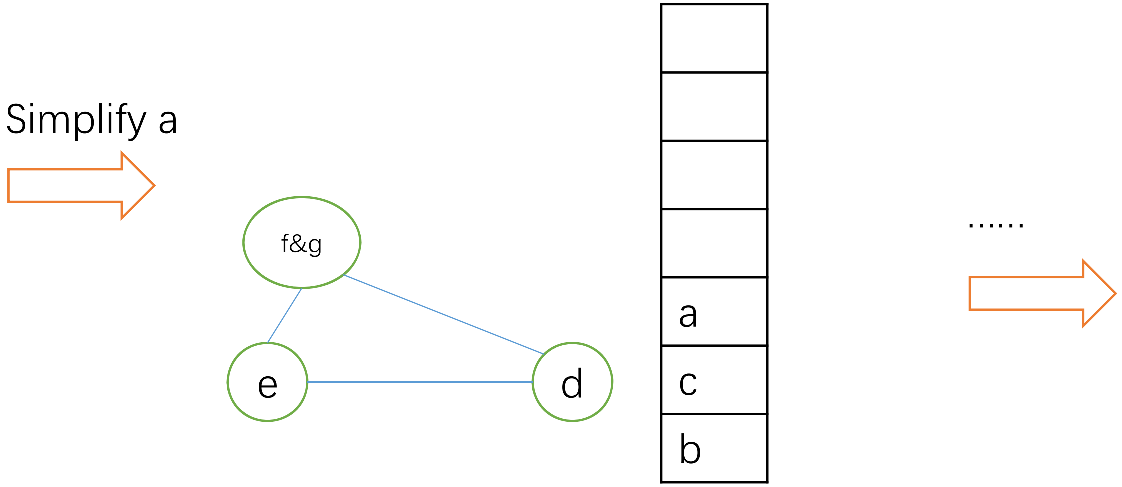
f is NOT spilled.

b.

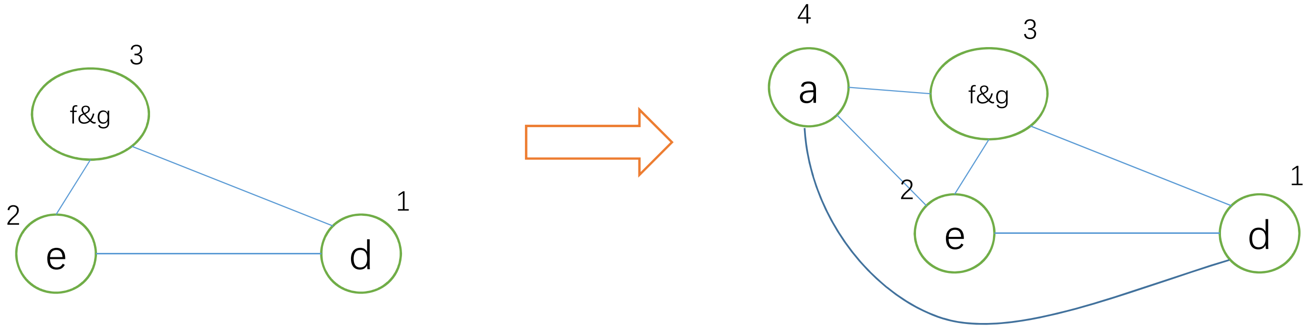


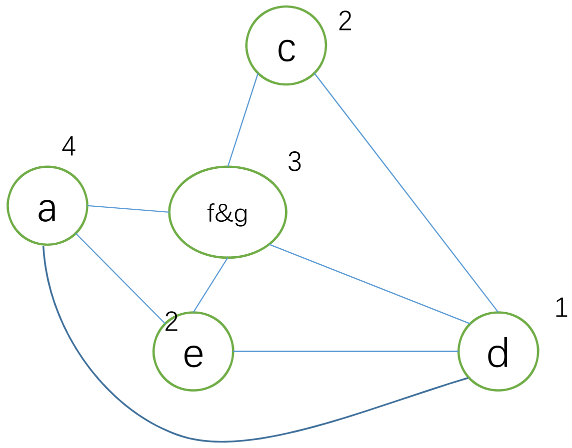
b
c
a
f
g
e
d
Coalescing f and g
with Briggs criterion
b
c
a
f&g
e
d
f&g
Simplify b
c
a
f&g
e
d
b
Simplify c
a
f&g
e
d
c
b

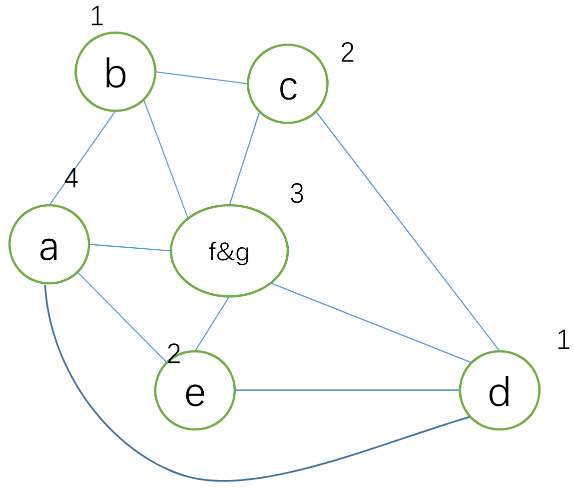
Simplify a



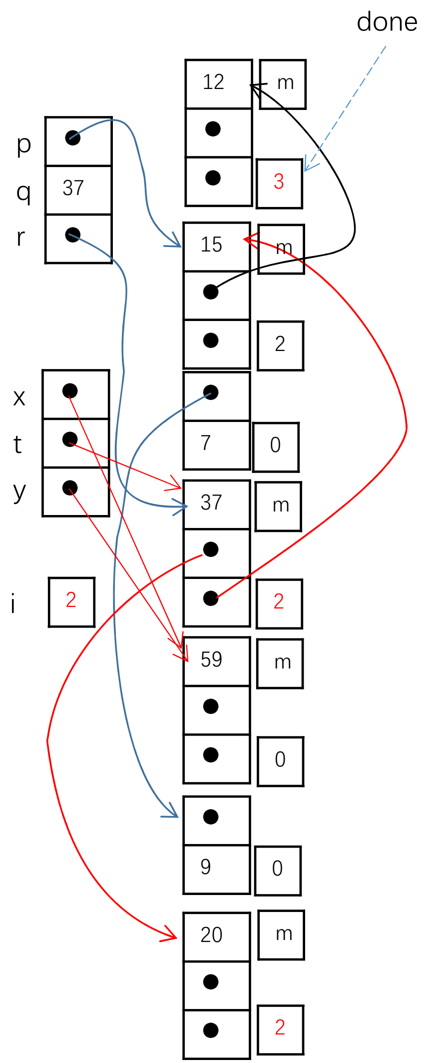
Select



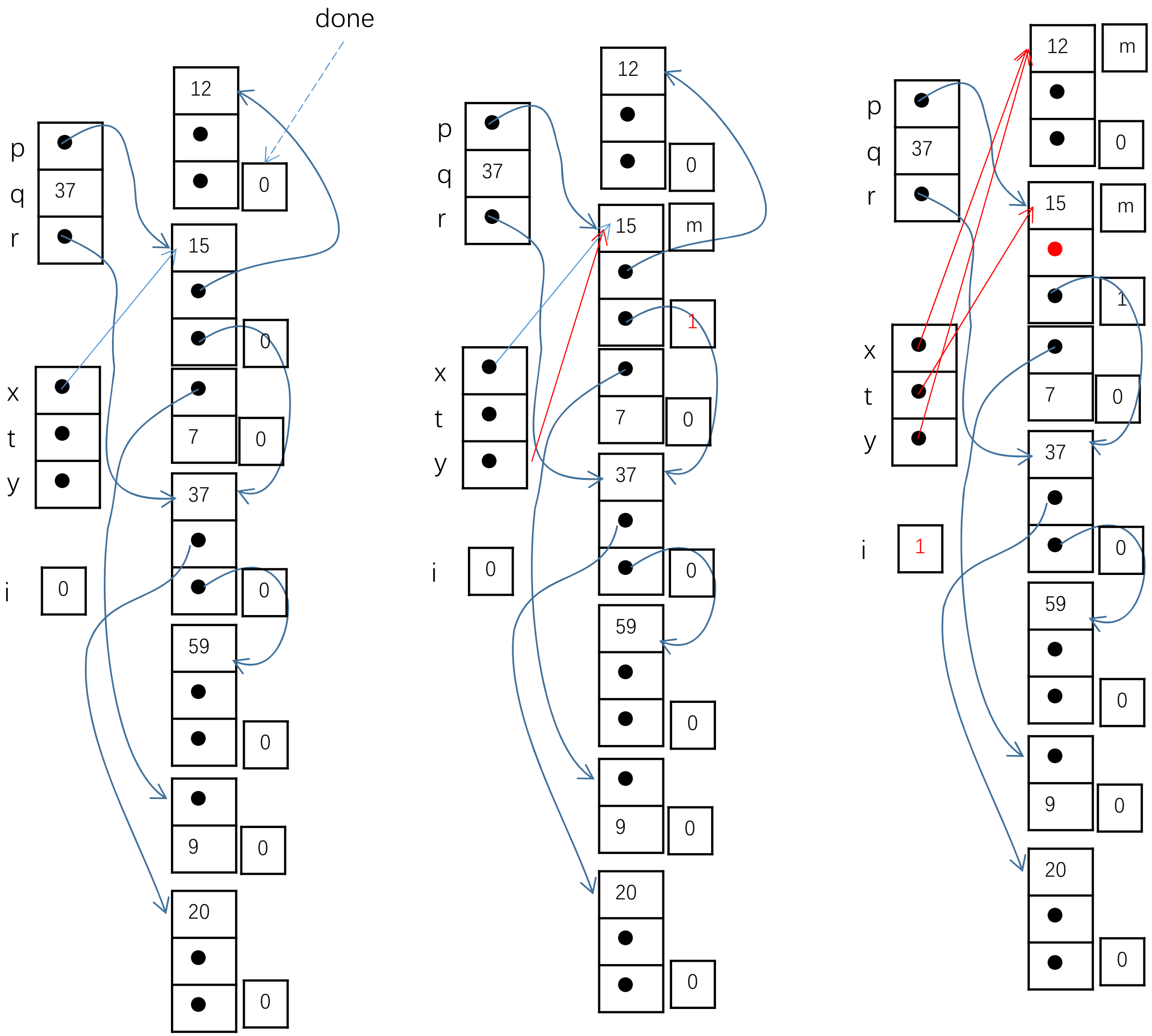

c
2
4
3
a
f&g
2
e
d
1


1
b
c
2
4
3
a
f&g
2
e
d
1

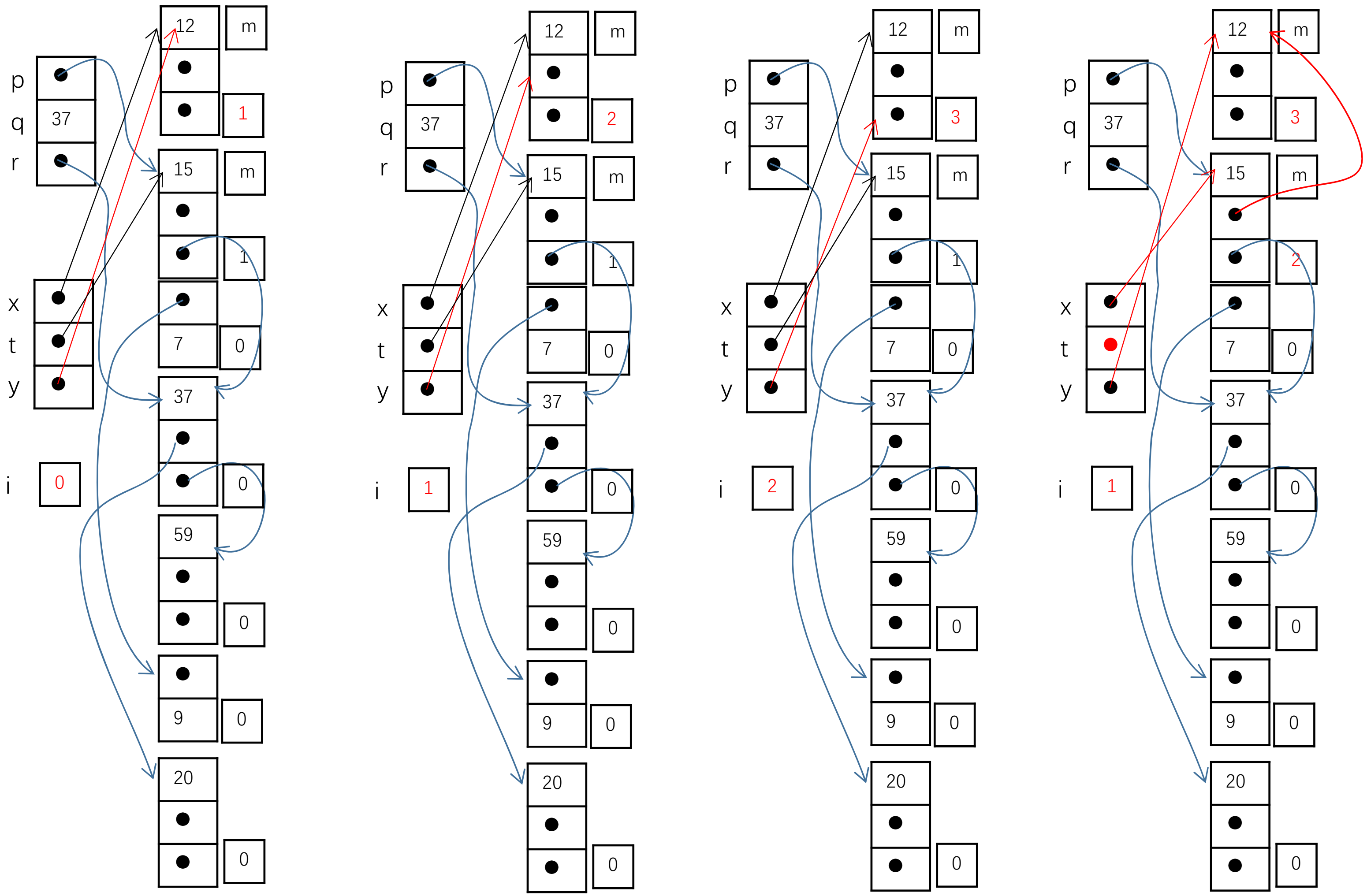
13.2 Solution:
When the node containing 59 is first marked, the state of the heap, the done flags, and variables t, x, and y are shown as the right figure.



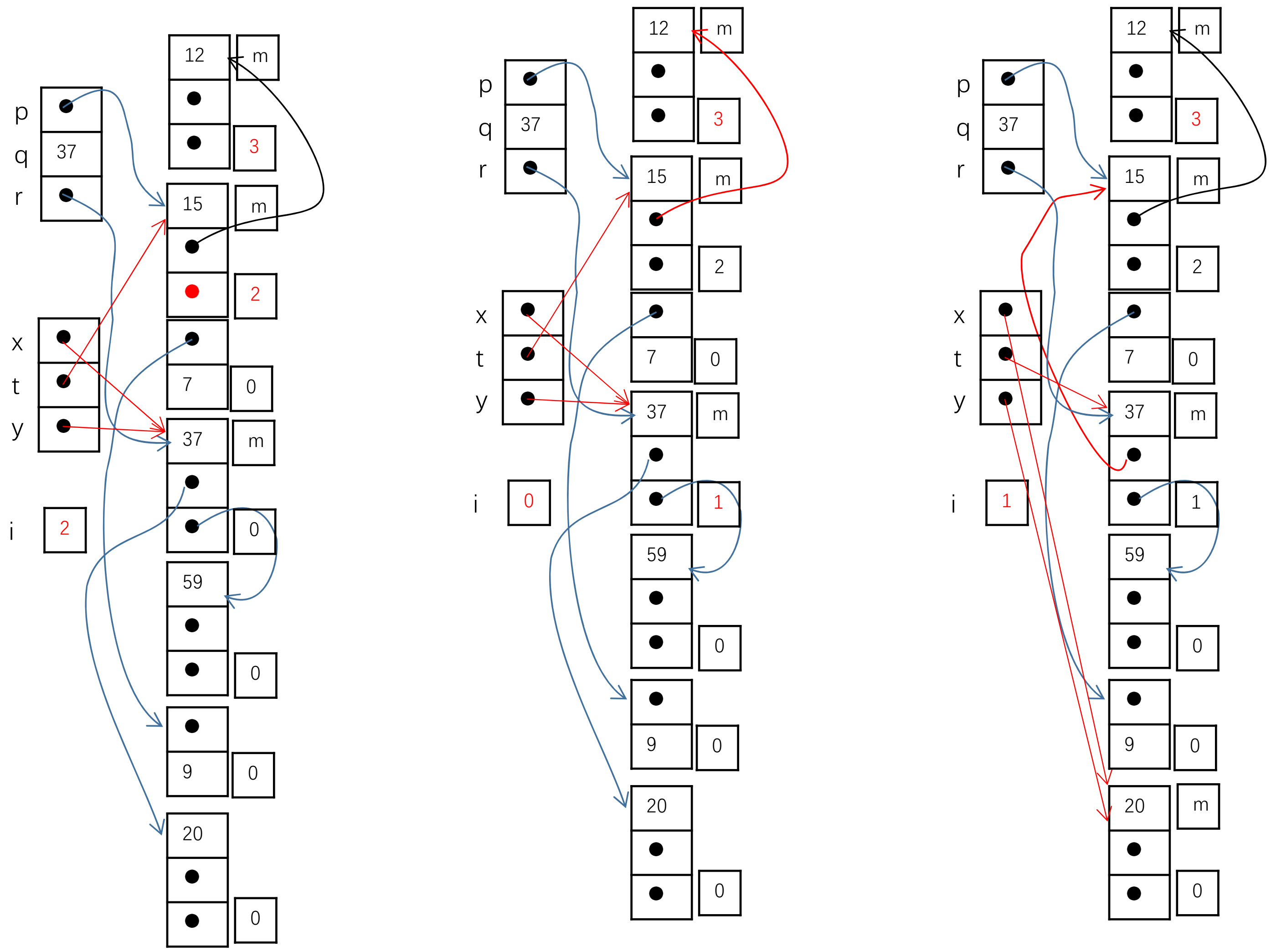
The following pages show the states at each iteration.



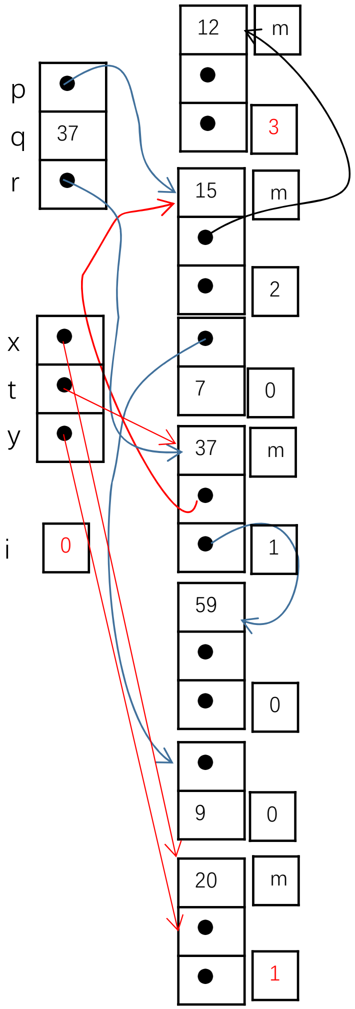
done
p
q
r
x
t
y
i
12
15
7
37
59
9
20
m

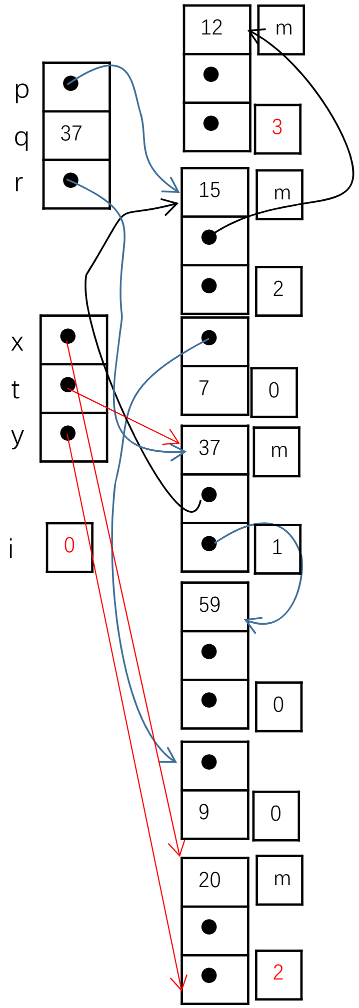


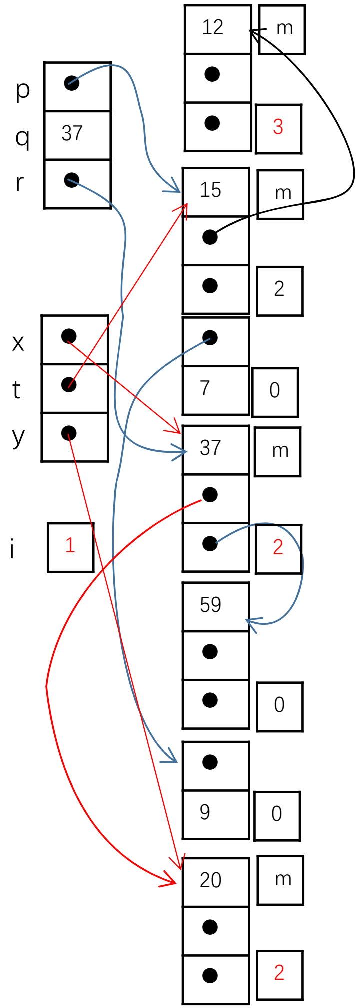
p
q
37
r
x
t
y
i
0
12
m
1
15
m
1
7
0
37
0
59
0
9
0
20
0
p
q
37
r
x
t
y
i
1
12
m
2
15
m
1
7
0
37
0
59
0
9
0
20
0
p
q
37
r
x
t
y
i
2
12
m
3
15
m
1
7
0
37
0
59
0
9
0
20
0
p
q
37
r
x
t
y
i
1
12
m
3
15
m
2
7
0
37
0
59
0
9
0
20
0

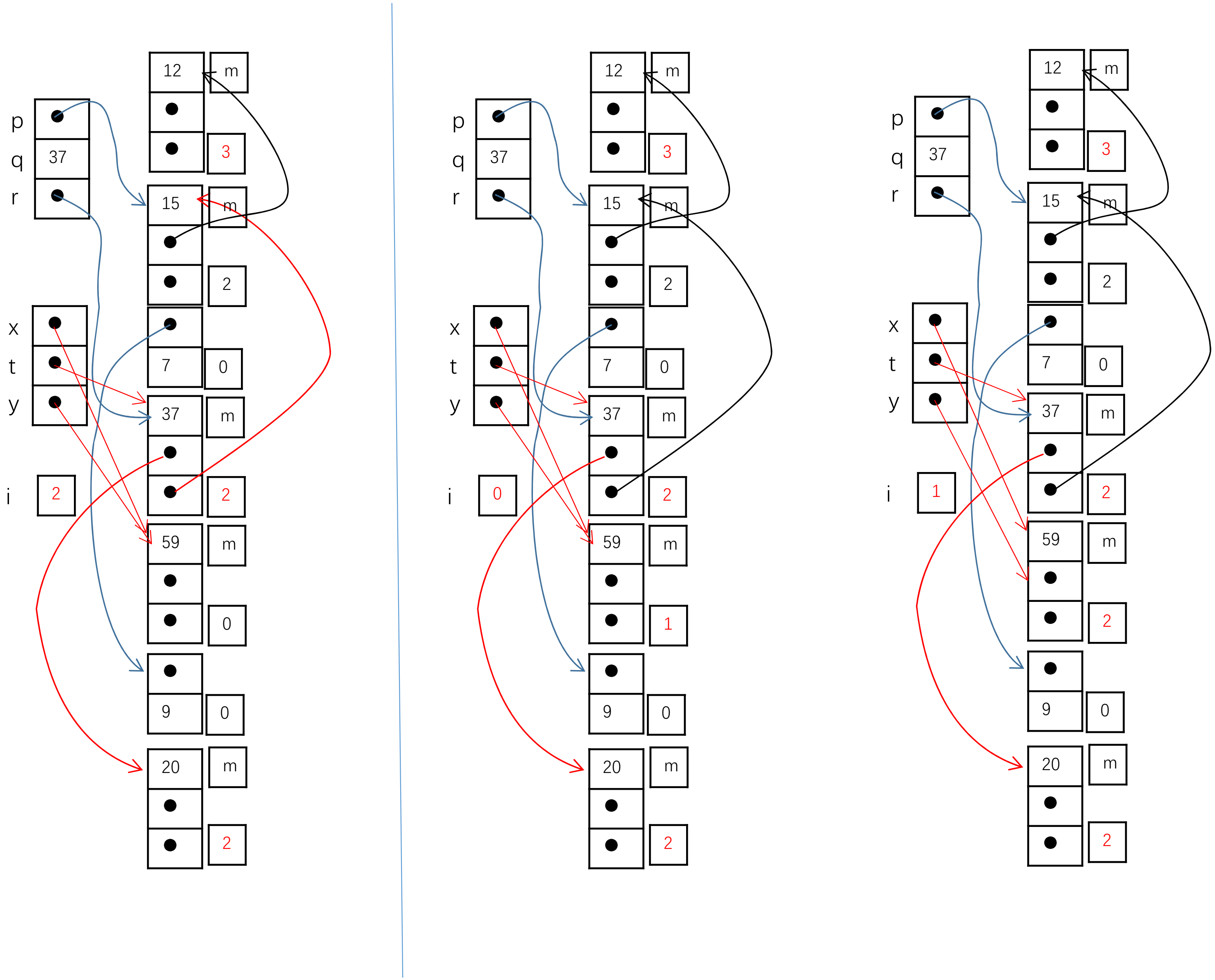


p
q
r
37
x
t
y
i
2
12
m
3
15
m
2
7
0
37
m
0
59
0
9
0
20
0
p
q
r
37
x
t
y
i
0
12
m
3
15
m
2
7
0
37
m
1
59
0
9
0
20
0
p
q
r
37
x
t
y
i
1
12
m
3
15
m
2
7
0
37
m
1
59
0
9
0
20
m
0

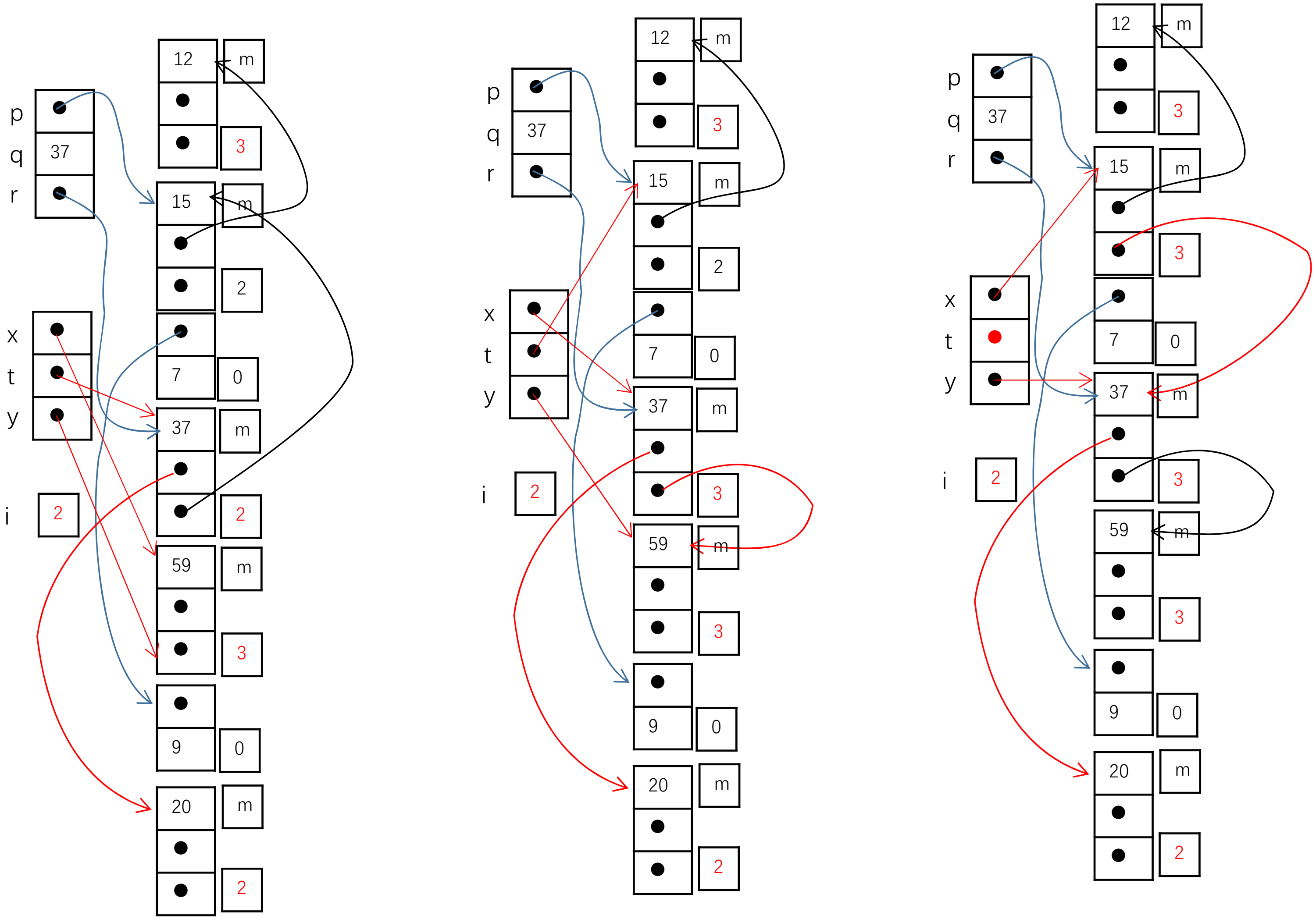

p
q
r
37
12
m
3
15
m
2
x
t
y
7
0
37
m
i
0
1
59
0
9
0
20
m
1


p
q
r
37
12
m
3
15
m
2
x
t
y
7
0
37
m
i
0
1
59
0
9
0
20
m
2

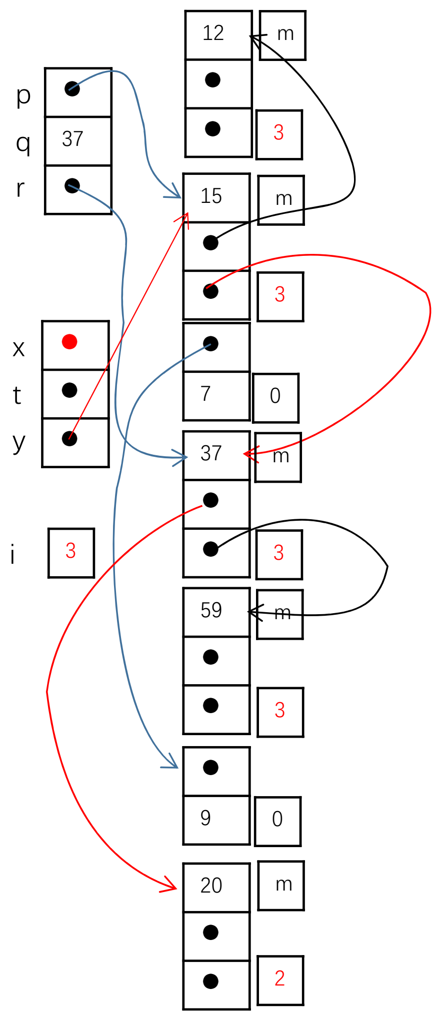

p
q
r
37
12
m
3
15
m
2
x
t
y
7
0
37
m
i
1
2
59
0
9
0
20
m
2



p
q
r
37
x
t
y
i
2
12
m
3
15
m
2
7
0
37
m
2
59
m
0
9
0
20
m
2
p
q
r
37
x
t
y
i
0
12
m
3
15
m
2
7
0
37
m
2
59
m
1
9
0
20
m
2
p
q
r
37
x
t
y
i
1
12
m
3
15
m
2
7
0
37
m
2
59
m
2
9
0
20
m
2

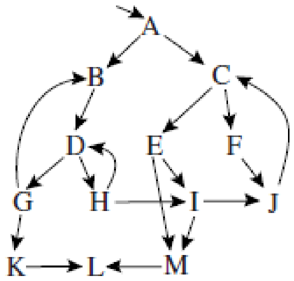


p
q
r
37
x
t
y
i
2
12
m
3
15
m
2
7
0
37
m
2
59
m
3
9
0
20
m
2
p
q
r
37
x
t
y
i
2
12
m
3
15
m
2
7
0
37
m
3
59
m
3
9
0
20
m
2
p
q
r
37
x
t
y
i
2
12
m
3
15
m
3
7
0
37
m
3
59
m
3
9
0
20
m
2



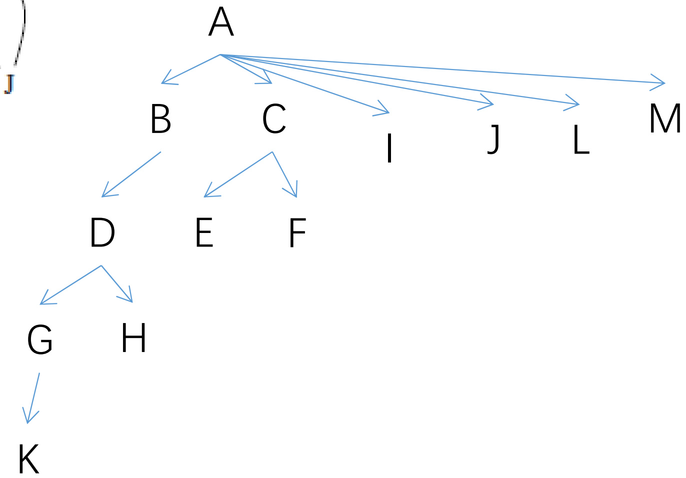
p
q
r
37
x
t
y
i
3
12
m
3
15
m
3
7
0
37
m
3
59
m
3
9
0
20
m
2

18.1 a. The dominators are listed as follows.



D(A) = {A}
D(B) = {B} U (D(A) ∩ D(G)) = {B, A}
D(D) = {D} U (D(B) ∩ D(H)) = {D, B, A}
D(G) = {G, D, B, A}
D(H) = {H, D, B, A}
D(K) = {K} U {G} = {K, G, D, B, A}
D(C) = {C, A}
D(E) = {E, C, A}
D(F) = {F, C, A}
D(I) = {I} U (D(E) ∩ D(H)) = {I, A}
D(J) = {J} U (D(I) ∩ D(F)) = {J, A}
D(M) = {M} U (D(E) ∩ D(I)) = {M, A}
D(L) = {L} U (D(K) ∩ D(M) = {K, A}

b. The dominator tree is show as follows.



c. There are two natural loops:

{B, D, G}

{D, H}

18.2 a. The graph of Figure 2.8.

The nodes are renumbered as shown in the figure. The immediate-dominators are listed in the 3rd column of the following table.

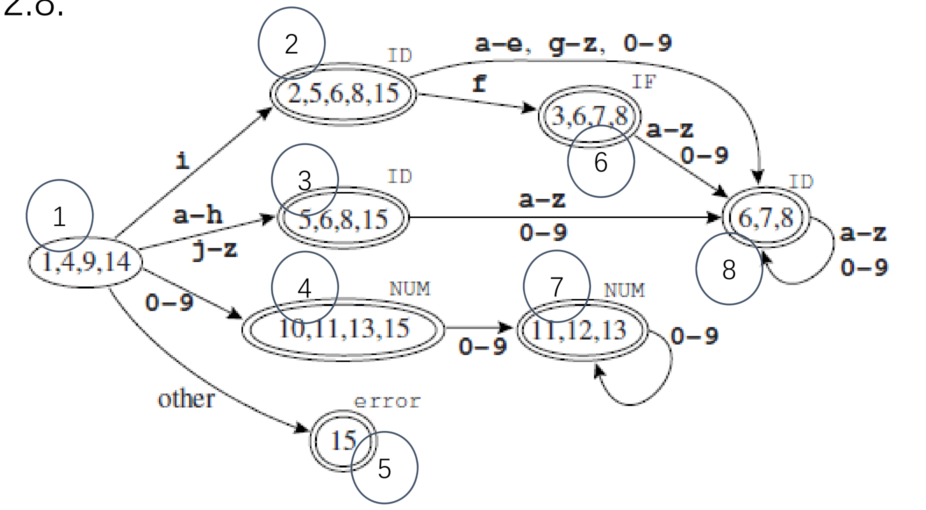
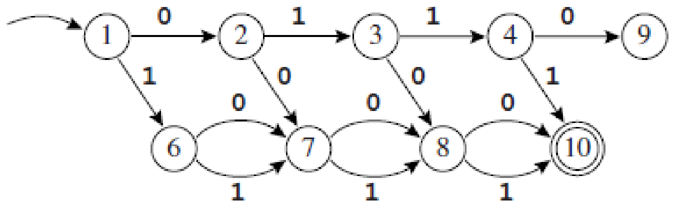


FIGURE 2.8. NFA converted to DFA.

	Dominators	immediate-dominator
1	1	
2	2, 1	1
3	3, 1	1
4	4, 1	1
5	5, 1	1
6	6, 2, 1	2
7	7, 4, 1	4
8	8, 1	1

b. The graph of Exercise 2.3a.

a.

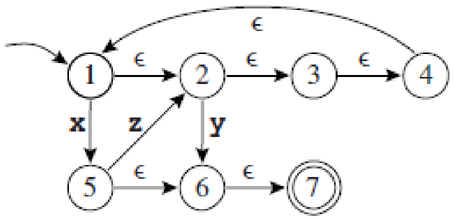


The immediate-dominators are listed in the 3rd column of the following table.

	Dominators	immediate-dominator
1	1,	
2	1, 2	1
3	1, 2, 3	2
4	1, 2, 3 , 4	3
6	1, 6	1
7	1, 7	1
8	1, 8	1
9	1, 2, 3 , 4, 9	4
10	1, 10	1

c. The graph of Exercise 2.5a.

a.



The immediate-dominators are listed in the 3rd column of the following table.

	Dominators	immediate-dominator
1	1	
2	2, 1	1
3	1, 2, 3	2
4	1, 2, 3, 4	3
5	5, 1	1
6	6, 1	1
7	7, 6, 1	6

d. The graph of Figure 3.27.

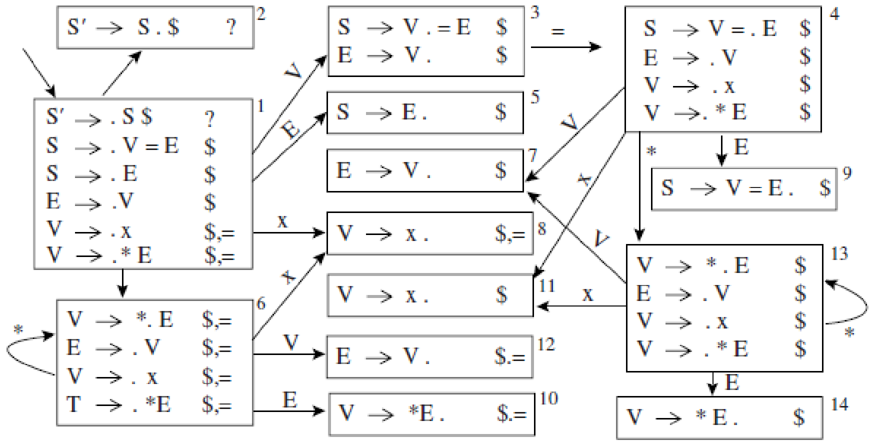


FIGURE 3.27. LR(1) states for Grammar 3.26.

The immediate-dominators are listed in the 3rd column of the following table.

	Dominators	immediate-dominator
1	1	
2	2, 1	1
3	3, 1	1
4	1, 3, 4	3
5	5, 1	1
6	6, 1	1
7	1, 3, 4, 7	4
8	1,	1
9	1, 3, 4, 9	4
10	1, 6, 10	6
11	1, 3, 4, 11	4
12	1, 6, 12	6
13	1, 3, 4, 13	4
14	1, 3, 4, 13, 14	13

18.6 Solution: suppose h1 is the node after the loop header h inside the loop. We insert a node hb between h and h1, with an edge h→hb. All the successors of h are disconnected from h, and become the successors of hb. An example is shown as follows.