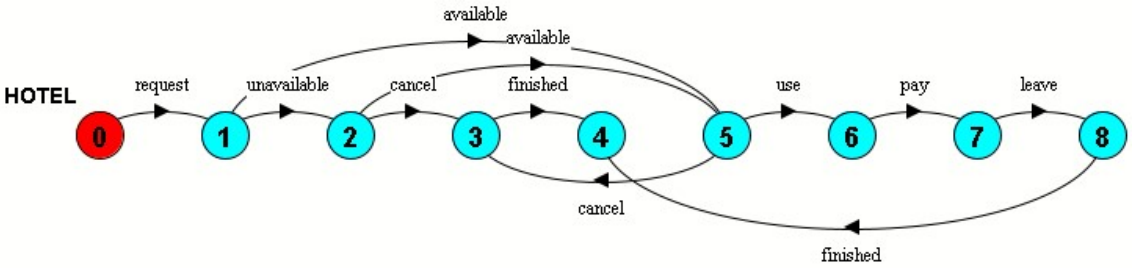
1.) Consider the following simple hotel reservation system. A customer makes a room request. If room is available, a confirmation is sent to the customer, otherwise the customer is put on a reservation list. If a room is confirmed, the customer may either use it, pay for the room, leave and the whole transaction is archived. However, the customer may also cancel his/her reservation. When the customer is on waiting list, a room may become available, and then a confirmation is sent to a customer. The customer may also give up waiting and cancel his/her request.

Model this reservation system as a FSP process reservation. Note that this process always stops, so you must use the process STOP. Also provide appropriate labelled transition system (use LTSA).




```
HOTEL = (request -> REQUESTED)
REQUESTED = (available -> CONFIRMED | unavailable -> WAITLIST)
CONFIRMED = (use -> pay -> leave -> ARCHIVE | cancel -> CANCELLED)
WAITLIST = (available -> CONFIRMED | cancel -> CANCELLED)
ARCHIVE = (finished -> STOP)
CANCELLED = (finished -> STOP)
```

2.a) For each one of the following three processes, give the Finite State Processes (FSP) description of the labelled transition graph. Dots indicate initial states.

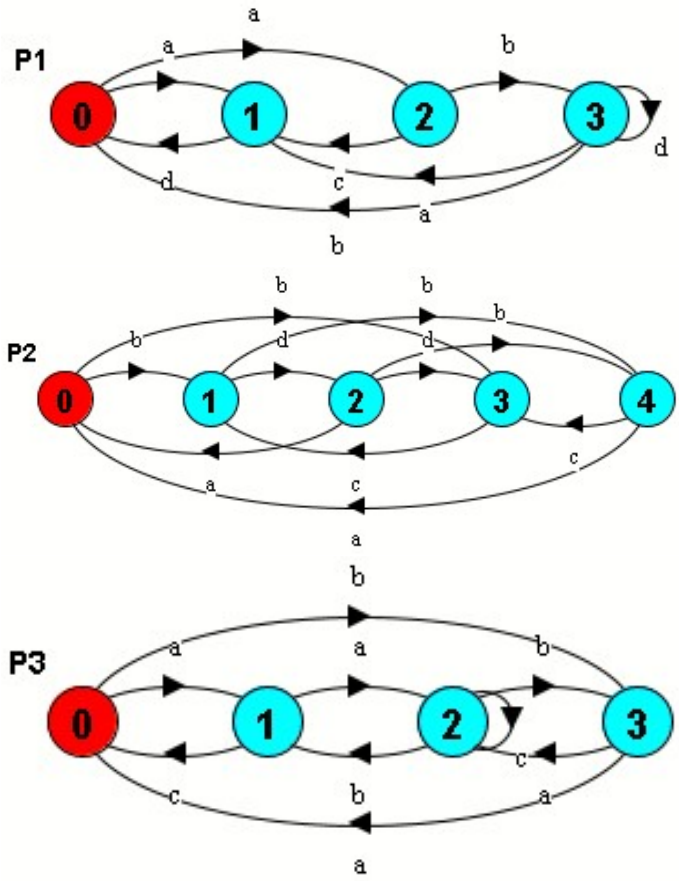
```
P1 = A
A = (a -> B | a -> D)
B = (b -> C | c -> D)
C = (d -> C | a -> D | b -> A)
D = (d -> A)

P2 = A
A = (b -> C | b -> B)
B = (b -> E | d -> D)
C = (c -> B)
D = (a -> A | b -> E | d -> C)
E = (c -> C |a -> A)

P3 = A
```

```
A = (b -> B | a -> D)
B = (a -> A | a -> C)
C = (c -> C | b -> B | b -> D)
D = (a -> C | c -> A)
```

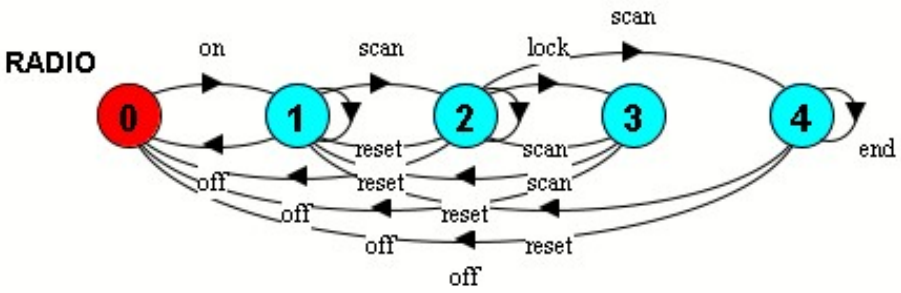
2.b) Use LTSA to transform the solutions to 2.a back into labelled transition systems. Compare the results and discuss differences (if any).



- processes P1 and P2 share the same set of transitions/labels (i.e. $\{a, b, c, d\}$)
  - P3 has a similar set of transitions/labels as P1 and P2 with the exception of transition d. (i.e. $\{a, b, c\}$)
- P1 and P3 share the same set of states (i.e. $\{A, B, C, D\}$)
  - P2 has a similar set of states as P1 and P2 with the exception of state E. (i.e. $\{A, B, C, D, E\}$)

3.) A miniature portable FM radio has three controls. An on/off switch turns the device on and off. Tuning is controlled by two buttons scan and reset which operate as follows. When the radio is turned on or reset is pressed, the radio is tuned to the top frequency of the FM band (108 MHz). When scan is pressed, the radio scans towards the bottom of the band (88 MHz). It stops scanning when it locks onto a station or it reaches the bottom (end). If the radio is currently tuned to a station and scan is pressed, then it start to scan from the frequency of that station towards the bottom. Similarly, when reset is pressed the receiver tunes to the top. Model the radio as a FSP process RADIO. Also provide an appropriate labelled transition system.

```
RADIO = OFF
OFF = (on -> TOP)
TOP = (off -> OFF | scan -> SCANNING | reset -> TOP)
SCANNING = (off -> OFF | scan -> BOTTOM |scan -> SCANNING | reset -> TOP |
    lock -> TUNING)
TUNING = (scan -> SCANNING | off -> OFF | reset -> TOP)
BOTTOM = (end -> BOTTOM | off -> OFF | reset -> TOP)
```

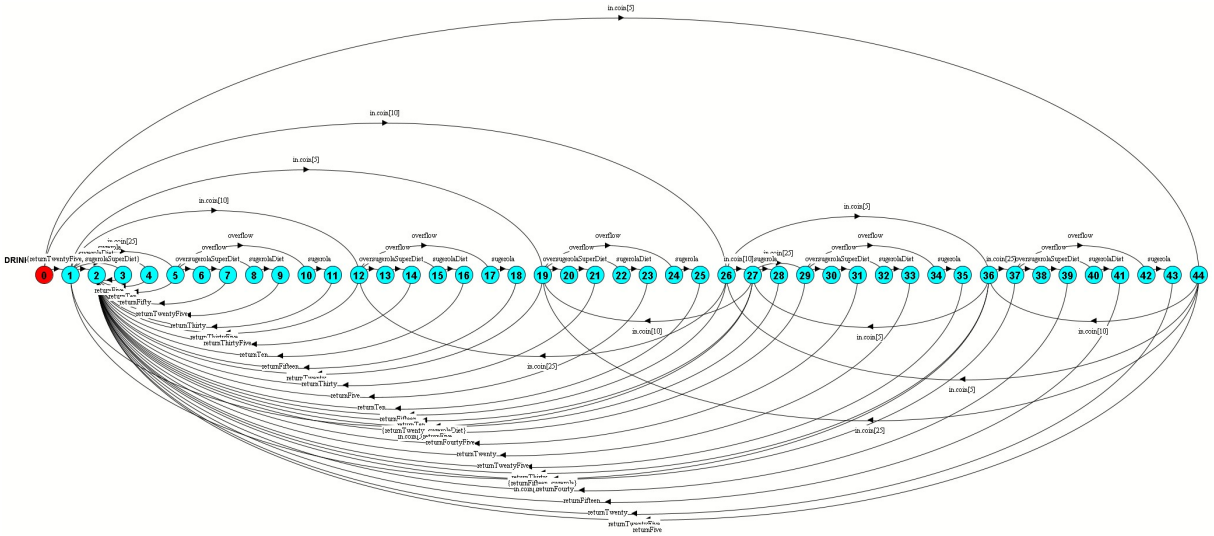


4.) A drinks dispending machine charges 15c for can of Sugerola, 20c for a can of SugerolaDiet and 25c for a can of SugerolaSuperDiet. The machine accepts coins with denominations 5c, 10c and 25c and gives changes. Model the machine as an FSP process, DRINKS.

The following assumptions were made:

- You can continue adding money into the vending machine until your balance is 30¢. Once you have a balance of 30¢, you cannot add more money because you can purchase everything with 30¢. The reason why you can still input money when you have a balance of 25¢is because the assignment hint file said we need states up to 50¢, and the only way to get 50¢is 2 25¢.
- You can only purchase one item per transaction.
- At the beginning of every state, there is a returnX → STOP because the person can cancel their order at anytime.

```
DRINKS = PAY
PAY = (in.coin[5] -> FIVE | in.coin[10] -> TEN | in.coin[25] -> TWENTYFIVE)
FIVE = (returnFive -> STOP | in.coin[5] -> TEN | in.coin[10] -> FIFTEEN |
    in.coin[25] -> THIRTY)
TEN = (returnTen -> STOP | in.coin[5] -> FIFTEEN | in.coin[10] -> TWENTY |
    in.coin[25] -> THIRTYFIVE)
FIFTEEN = (returnFifteen -> STOP | sugerola -> STOP | in.coin[5] -> TWENTY |
    in.coin[10] -> TWENTYFIVE | in.coin[25] -> FOURTY)
TWENTY = (returnTwenty -> STOP | sugerola -> returnFive -> STOP
    | sugerolaDiet -> STOP | in.coin[5] -> TWENTYFIVE |
    in.coin[10] -> THIRTY | in.coin[25] -> FOURTYFIVE)
TWENTYFIVE =(returnTwentyFive -> STOP | sugerola -> returnTen -> STOP
    | sugerolaDiet -> returnFive -> STOP | sugerolaSuperDiet -> STOP |
    in.coin[5] -> THIRTY | in.coin[10] -> THIRTYFIVE |
    in.coin[25] -> FIFTY)
THIRTY = (returnThirty -> STOP |
    overflow -> sugerola -> returnFifteen -> STOP |
    overflow -> sugerolaDiet -> returnTen -> STOP |
    overflow -> sugerolaSuperDiet -> returnFive -> STOP)
THIRTYFIVE = (returnThirtyFive -> STOP |
    overflow -> sugerola -> returnTwenty -> STOP |
    overflow -> sugerolaDiet -> returnFifteen -> STOP |
    overflow -> sugerolaSuperDiet -> returnTen -> STOP)
FOURTY = (returnFourty -> STOP |
    overflow -> sugerola -> returnTwentyFive -> STOP
    | overflow -> sugerolaDiet -> returnTwenty -> STOP |
    overflow -> sugerolaSuperDiet -> returnFifteen -> STOP)
FOURTYFIVE = (returnFourtyFive -> STOP |
    overflow -> sugerola -> returnThirty -> STOP |
    overflow -> sugerolaDiet -> returnTwentyFive -> STOP |
        overflow -> sugerolaSuperDiet -> returnTwenty -> STOP)
FIFTY = (returnFifty -> STOP |
    overflow -> sugerola -> returnThirtyFive -> STOP |
    overflow -> sugerolaDiet -> returnThirty -> STOP |
    overflow -> sugerolaSuperDiet -> returnTwentyFive -> STOP)
```

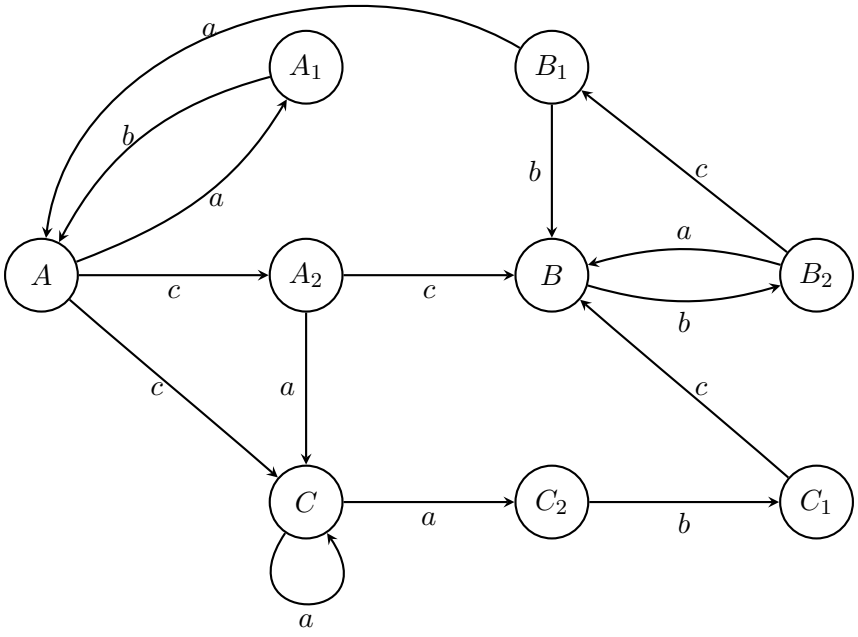

in.coin[5]
in.coin[10]
in.coin[5]
in.coin[10]
in.coin[5]
overflow
overflow
overflow
overflow
overflow
overflow
DRINK
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
returnFifty...
returnTwentyFive
returnThirty...
returnFive
returnTen...
returnFifteen...
returnTwenty...
returnFive
in.coin[10]
in.coin[5]
in.coin[10]
in.coin[25]
in.coin[5]
in.coin[25]
returnFourtyFive
returnTwenty...
returnTwentyFive
returnFifteen...
returnTwenty...
returnFive

6.a) Construct an equivalent Labelled Transition System using the rules from page 16 of Lecture Notes 2.

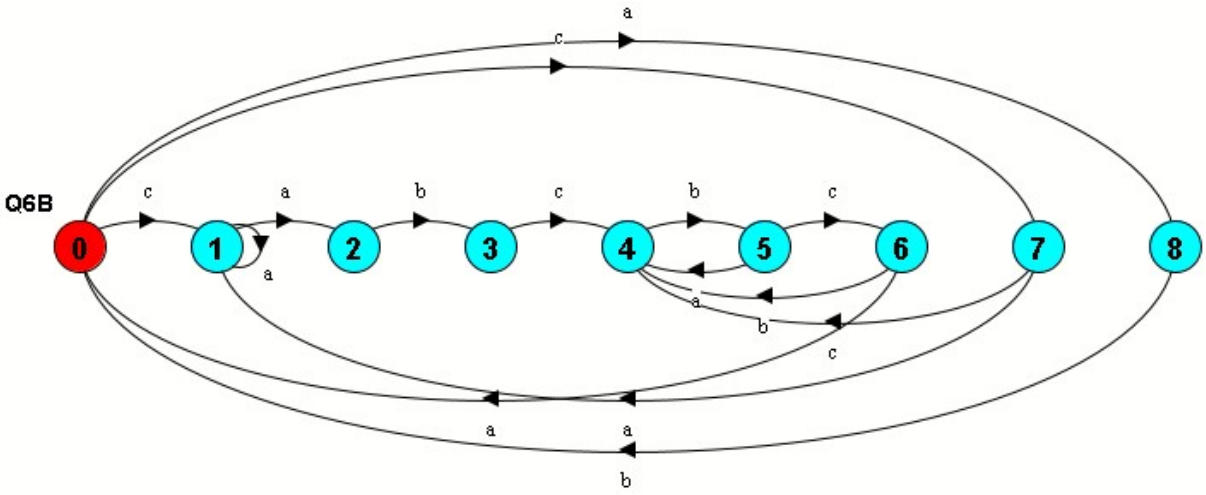
```
A = a -> A1 | c -> A2 | c -> C
    A1 = b -> A
    A2 = a -> C | c -> B

B = b -> B2
    B1 = a -> A | b -> B
    B2 = a -> B | c -> B1

C = a -> C2 | a -> C
    C1 = c -> B
    C2 = b -> C1
```



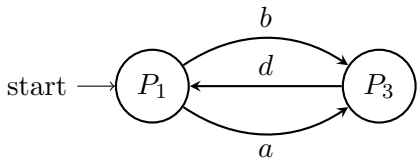
6.b) Use LTSA to derive appropriate LTS, and, if different then yours, analyse and explain differences.

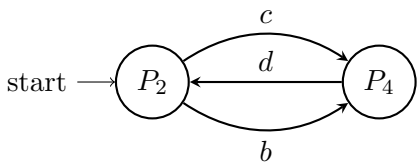


Both LTSA's are the same, minus the fact that you can't change state names in LTSA. (If you check Q6.lts, I inputted my constructed LTS into LTSA and it generated the same LTSA as well.)

8.) Model the net N1 as a composition of FSP processes

I converted the petri net to their separate LTS's:





Now, we can convert the LTS's into FPS.

```
P1 = (b -> P3 | a -> P3)
P3 = (d -> P1)

P2 = (c -> P4 | b -> P4)
P4 = (d -> P2)

||N1 = (P1 || P2)
```

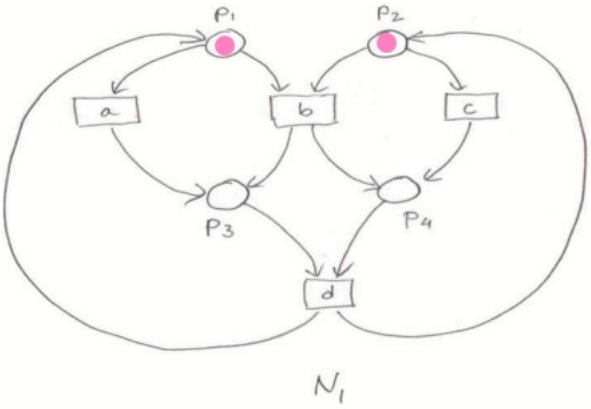
9.) Model the system from page 10 of Lecture Notes 3 as a composition of FSP processes. In this case, the entities that are represented by places in the Petri Nets model, must be represented by actions/transitions in FSP model.

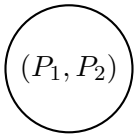
```
COMPUTER1 = (idle1 -> (read1 -> COMPUTER1 | write1 -> COMPUTER1))
COMPUTER2 = (idle2 -> (read2 -> COMPUTER2 | write2 -> COMPUTER2))
MUT = (write1 -> MUT | write2 -> MUT)

|| LOCKED = (COMPUTER1 || MUT || COMPUTER2)
```

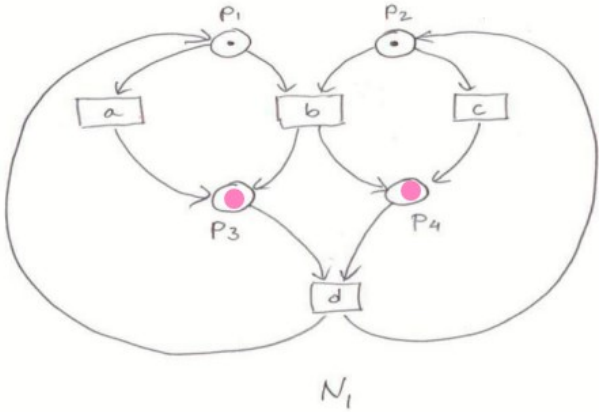
11.) Construct reachability graph (defined on page 18 of Lecture Notes 3) for the Petri net from Question 8.

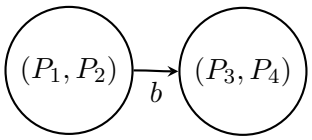
Initial state:



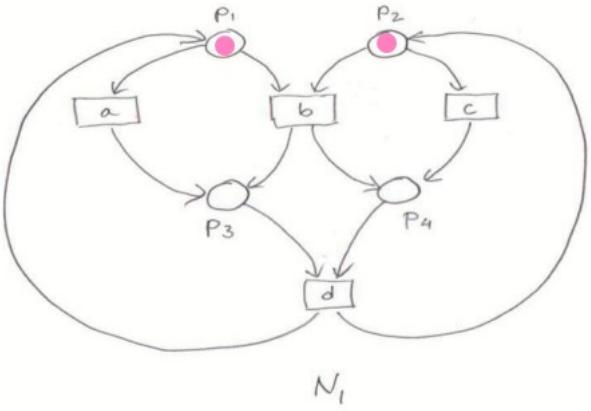


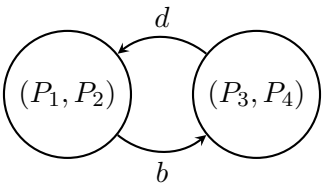
Firing b:



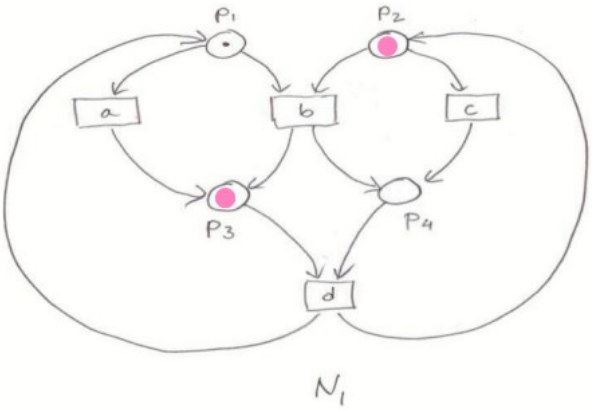


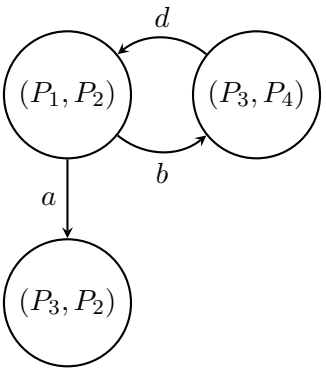
Firing d:



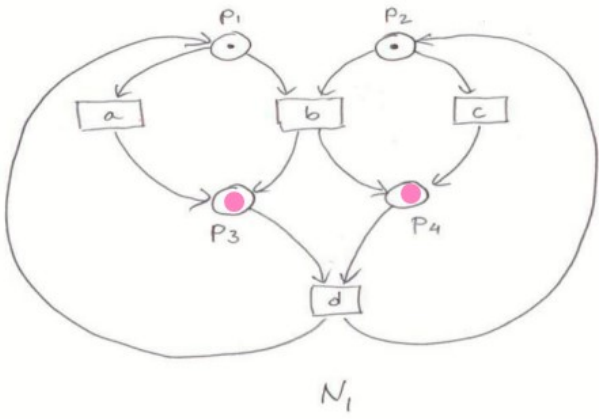


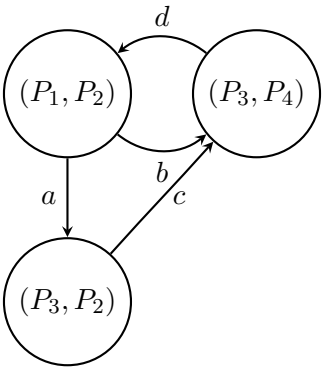
Firing a:



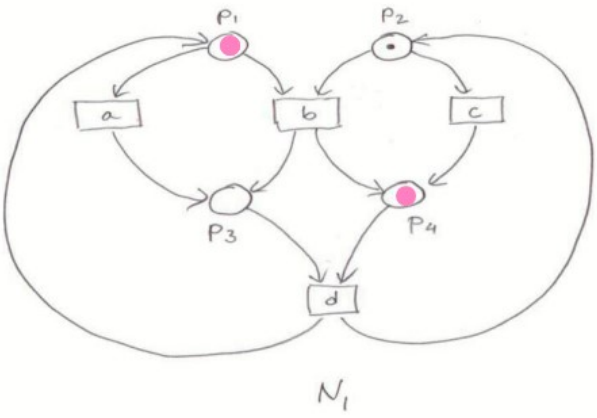


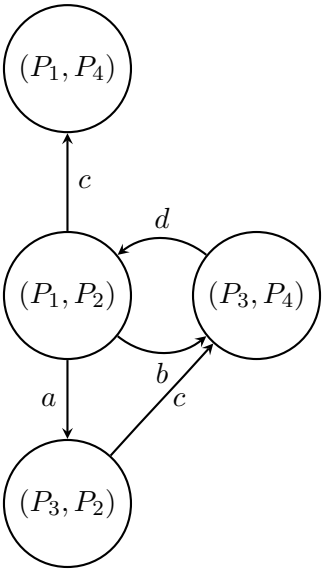
Firing c:



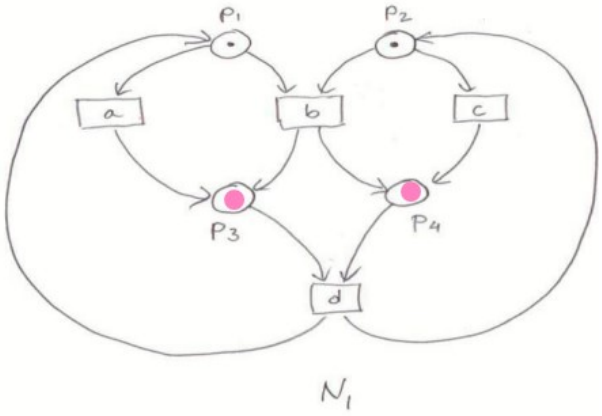


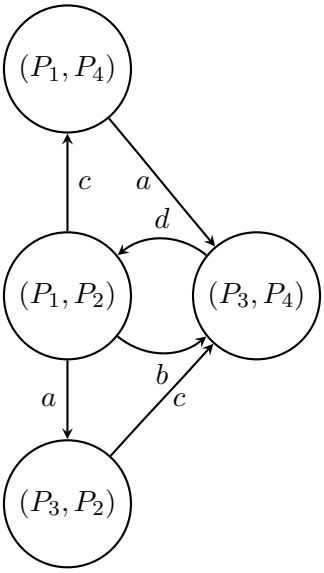
Back at state where we fire d. Return back to initial state.
Fire c:





Fire a:





Back at state where we fire d. All cases reached.

12.a) $P1 \approx P3$**, i.e. P2 and P2 are bisimilar**

Clearly, $p_0 \approx s_0$ because only the transition a can be executed in both cases. After the trace a, P_2 goes to the state p_1 and P_3 goes to the state s_1. $p_1 \approx s_1$ because only a can be executed in both cases. After the trace aa, P_2 goes to the state p_2 and P_3 goes to the state s_2. $p_2 \approx s_2$ because a, b, and c can be executed in all cases. After the trace aac, P_2 goes to the state p_3 and P_3 goes to the state s_3. $p_3 \approx s_3$ because only c and a can be executed in both cases. After the trace aaa or $aacc*$, P_2 goes to the state p_5 and P_3 goes to the state s_5. $p_5 \approx s_5$ because only c can be executed in both cases

12.b) $P1 \not\approx P2$**, i.e. P1 and P2 are not bisimilar**

Clearly $q_0 \approx p_0$ because only the transition a and b can be executed in both cases. After the trace a, P_1 goes to the state q_1 and P_2 goes to the state p_1. We can say that $q_1 \approx p_1$ because only a can be executed in both cases. After the trace aa, P_2 goes to the state p_2 and P_1 goes to either q_2 or q_3. However, $p_2 \not\approx q_2$ and $p_2 \not\approx q_3$. At p_2 a, b, and c can be executed, but at q_2 we can only execute c and a, and at q_3 we can only execute a and b. Hence, $P_1 \not\approx P_2$.

12.c) $P1 \not\approx P3$**, i.e. P1 and P3 are not bisimilar**

Clearly $q_0 \approx s_0$ because only the transition a and b can be executed in both cases. After the trace a, P_1 goes to the state q_1 and P_3 goes to the state s_1. We can say that $q_1 \approx s_1$ because only a can be executed in both cases. After the trace aa, P_3 goes to the state s_2 and P_1 goes to either q_2 or q_3. However, $s_2 \not\approx q_2$ and $s_2 \not\approx q_3$. At s_2 a, b, and c can be executed, but at q_2 we can only execute c and a, and at q_3 we can only execute a and b. Hence, $P_1 \not\approx P_3$.