CS 2SD3. Sample solutions to the assignment 1.

Total of this assignment is 134 pts. Each assignment is worth 10% of total. Most of solutions are not unique.

If you think your solution has been marked wrongly, write a short memo stating where marking in wrong and what you think is right, and resubmit to me via e-mail (as pdf). The deadline for a complaint is 2 weeks after the assignment is marked and returned.

1.[10] Model the following Road Deicing protocol as FSP. The road could be in one of the following states: Predicted Safe For Use, Predicted Unsafe For Traffic But Open, Closed. If road is ‘Predicted Safe For Use’, coming ‘Predicted Ice Formation’ changes its status to ‘Predicted Unsafe For Traffic But Open’. If road is ‘Predicted Unsafe For Traffic But Open’, ice may melt (i.e. action ‘Ice melts’ occurs) and the road is again ‘Predicted Safe For Use’, or it becomes unsafe for use (action ‘Unsafe for Use’) and it is in the state ‘Closed’, or it is treated (action ‘Road treated’) and it is in the state ‘Predicted Safe For Use’ again. If the road is ‘Closed’, either ‘Ice melts’ or it is treated (action ‘Road treated’), in both cases it becomes ‘Predicted Safe For Use’.

Provide an appropriate Labelled Transition System (use LTSA).

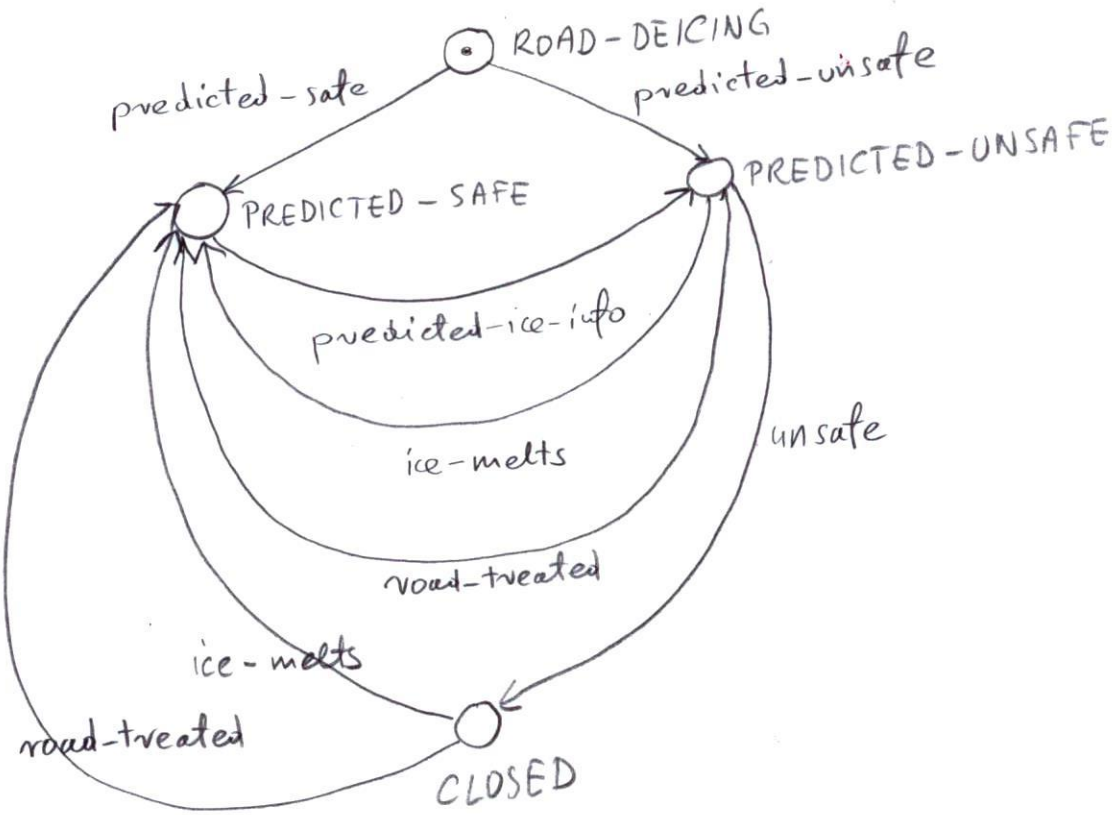
Hint: The processes: ROAD-DEICING, PREDICTED-SAFE, PREDICTED-UNSAFE, CLOSED.

Solution:

```
ROAD-DEICING = ( predicted-safe -> PREDICTED-SAFE |
                 predicted-unsafe -> PREDICTED-UNSAFE )
PREDICTED-SAFE = ( predicted-ice-info -> PREDICTED-UNSAFE)
PREDICTED-UNSAFE = (ice-melts - > PREDICTED-SAFE |
                    road-treated - > PREDICTED-SAFE |
                    unsafe-for-use -> CLOSED)
CLOSED = (ice-melts - > PREDICTED-SAFE | road-treated - > PREDICTED-SAFE )
```

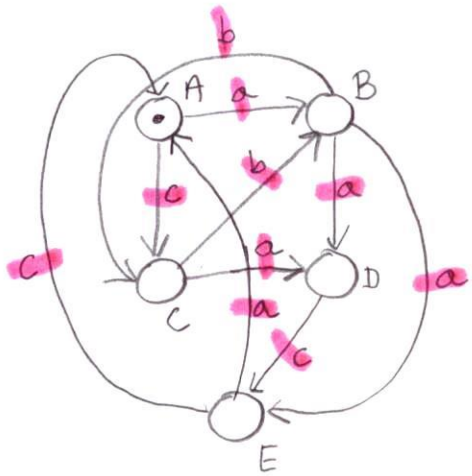
```
ROADDEICING = (predictedsafe -> PREDICTEDSAFE | predictedunsafe -> PREDICTEDUNSAFE),
PREDICTEDSAFE = ( predictediceinfo -> PREDICTEDUNSAFE),
PREDICTEDUNSAFE = (icemelts -> PREDICTEDSAFE | roadtreated -> PREDICTEDSAFE |
                   unsafeforuse -> CLOSED),
CLOSED = (icemelts -> PREDICTEDSAFE | roadtreated -> PREDICTEDSAFE ).
```

Expected LTS (LTSA should provide an equivalent one):

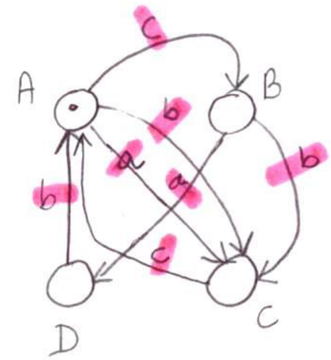


2.[15] a.[9] For each one of the following three processes, give the Finite State Processes (FSP) description of the labelled transition graph. Dots indicate initial states.

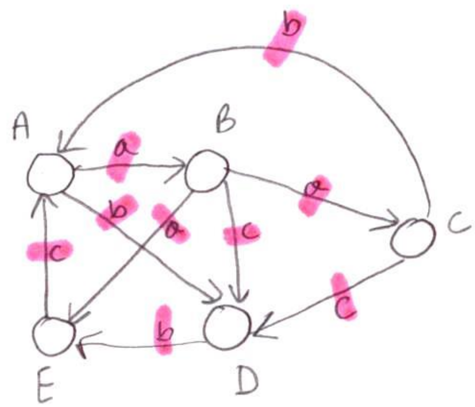
b.[6] Use LTSA to transform the solutions to 1.a back into labelled transition systems. Compare the results and discuss differences (if any).



P_1



P_2



P_3

a. P1 = A
A = (a → B | c → C)
B = (a → D | a → E | b → C)
C = (a → D | b → B)
D = (c → E)
E = (c → A)

or, P1 = A
A = (a → B | c → C)
B = (a → c → c → A | a → c → A | b → C)
C = (a → c → c → A | b → B)

P2 = A
A = (a → C | b → C | c → B)
B = (a → D | b → C)
C = (c → A)
D = (b → A)

or, P2 = A
A = (a → c → A | b → c → A | c → B)
B = (a → b → A | b → c → A)

P3 = A
A = (a → B | b → D)
B = (a → C | a → E | c → D)
C = (c → D | b → A)
D = (b → E)
E = (c → A)

or, P3 = A
A = (a → B | b → b → c → A)
B = (a → C | a → E | c → b → c → A)
C = (c → b → c → A | b → b → c → A)

b. Not done here.

3.[15] Consider the following set of FSPs:

A = (a → (b → C)) | (b → (a → D | c → A)) | c → B)
B = (b → (a → A | c → (b → C | a → D)))
C = (a → (b → (c → B)))
D = (c → A | c → (b → B))

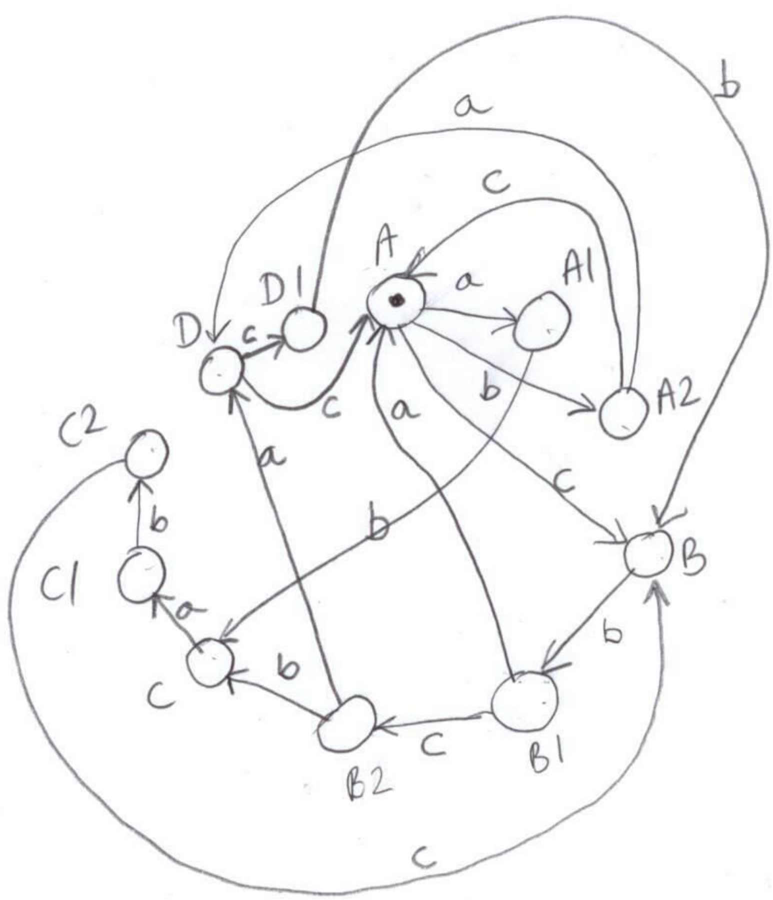
a.[12] Construct an equivalent Labelled Transition System using the rules from page 16 of Lecture Notes 2.

b.[3] Use LTSA to derive appropriate LTS, and, if different than yours, analyse and explain differences.

Solution: We first transform the equations to fit the patterns from page 16 of LN 2.

A = (a → A1 | b → A2 | c → B)
A1 = (b → C)
A2 = (a → D | c → A)
B = (b → B1)
B1= (a → A | c → B2)
B2 = (b → C | a → D)
C = (a → C1)
C1 = (b → C2)
C2 = (c → B)
D = (c → A | c → D1)
D1 = (b → B)

Now we can construct appropriate Labelled Transition System:



The one produced by LTSA should be equivalent.

4.[8] A sensor measures the water *level* of tank. The level (initially 5) is measured in units 0 ... 11. The sensor outputs a *low-danger* signal if the level is less than 2, a *low* signal if the level is 2 or 3, a *high* signal if the level is 8 or 9, and a *high-danger* signal if the level is more than 9; otherwise it outputs *normal*. Model the sensor as an FSP process *SENSOR* (this process is intended to be a part of bigger system that is not a subject of this question).
Hint: The alphabet of *SENSOR* is { *level*[0..9], *high-danger*, *high*, *low*, *low-danger*, *normal* }

Solution:

```
range Level = 0..11

SENSOR           = SENSOR[5],
SENSOR[w:Level] = (level[i:Level]  -> SENSOR[i]
                   | when (w<2) low-danger -> SENSOR[w]
                   | when (w==2 || w==3 ) low -> SENSOR[w]
                   | when (w==8 || w==9) high -> SENSOR[w]
                   | when (w>9) high-danger -> SENSOR[w]
                   | when (w>=4 && w<=7) normal -> SENSOR[w]
                  ).
```

5.[10] 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 stop 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.

Hint: The alphabet of RADIO is {on, off, scan, reset, lock, end}.

Solution:

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

6.[18] a.[8] Show that the processes ||S1 and S2 generate the same Labelled Transition Systems, i.e. LTS(||S1) = LTS(S2) (or equivalently, they generate the same behaviour)

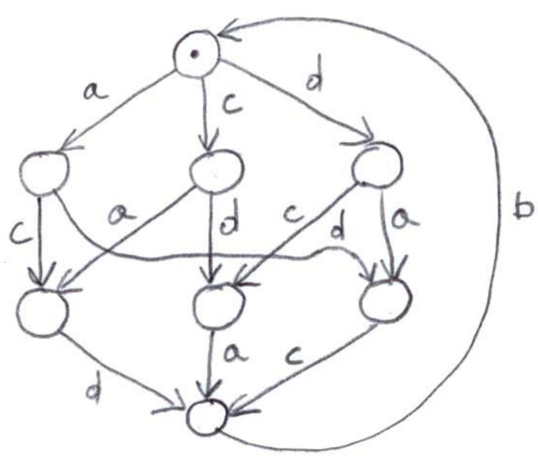
P = (a → b → P).
Q = (c → b → Q).
R = (d → b → R).
||S1 = (P || Q || R).

S2 = (a → c → d → b → S2 | a → d → c → b → S2 |
c → a → d → b → S2 | c → d → a → b → S2 |
d → a → c → b → S2 | d → c → a → b → S2).

b.[10] Using a method presented on page 17 of Lecture Notes 3 and pages 10-11 of Lecture Notes 4, transform the processes $\|S1$ and $S2$ into appropriate Petri nets. Are these nets identical? Explain the difference. Which one allows *simultaneity*?

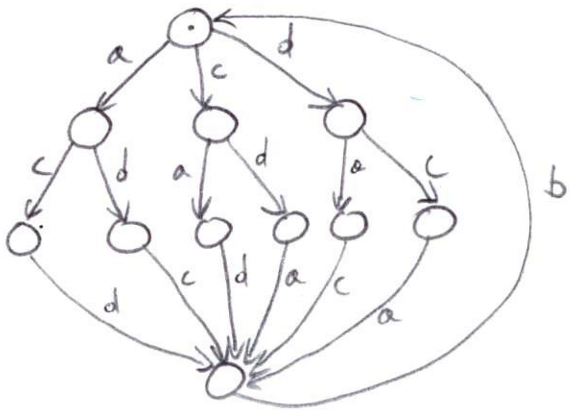
Solution:

a. Note that ||S1 does not involve non-determinism and S2 does not involve the operator '||'. Hence we do not need bisimulation, just to show that Traces(LTS(||S1)) = Traces(LTS(S2)) is sufficient. The 'best', i.e. minimum state, Labelled Transition System corresponding to ||S1, i.e. LTS(||S1) is shown below.



LTS(||S1)

However, some other equivalent, not necessary minimum state, solutions are possible, for example LTS_1(||S1) shown below, which some may see as more intuitive one.

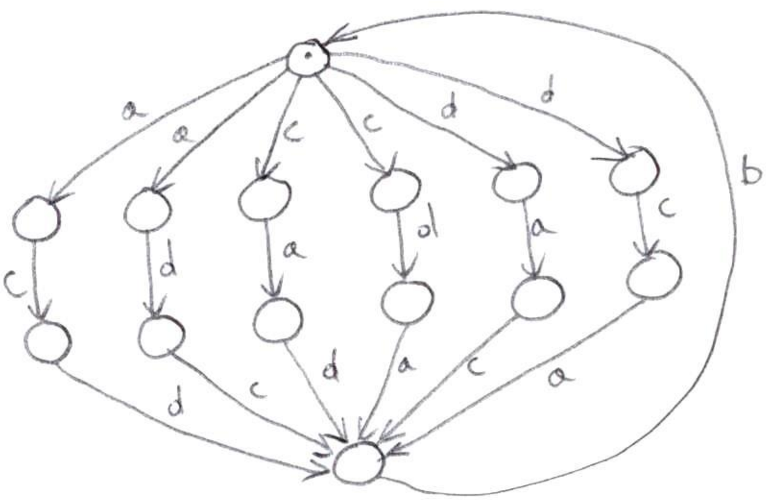


$LTS_1(\|S1)$

Clearly Traces(LTS(||S1)) = Traces(LTS_1(||S1)) = Pref(((acd $\cup$ adc $\cup$ cad $\cup$ cda $\cup$ dac $\cup$ dca)b)*).

The straightforward Labelled Transition System corresponding to S2, say, LTS_1(S2) is shown below:

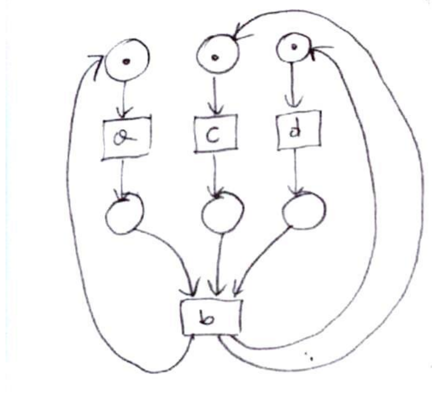
$LTS_1(S2) =$



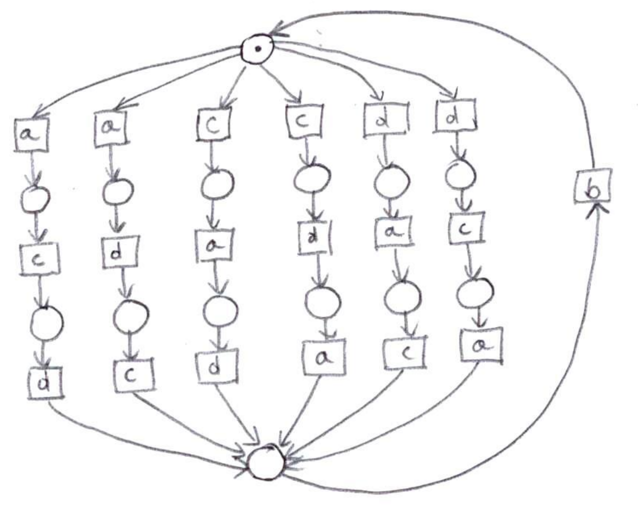
Clearly LTS_1(S2) is non-deterministic finite automaton, but after standard transformation it into a deterministic one (‘gluing together two a’s, two c’s and two d’s, see a relevant course on finite automata), we get exactly LTS_1(||S1). When only traces are considered, LTS(||S1) and LTS_1(S2) are equivalent as Traces(LTS(||S1)) = LTS_1(||S1) = Traces(LTS_1(S2)).

b. The Petri Nets corresponding to ||S1 and S2 are different. Let $N_{\|S1}$ be a net corresponding to ||S1, and N_{S2}, be a net corresponding to S2.

The net $N_{\|S1}$ allows simultaneity, all three a, c, d can be executed simultaneously, and each two of these three also can be executed simultaneously.

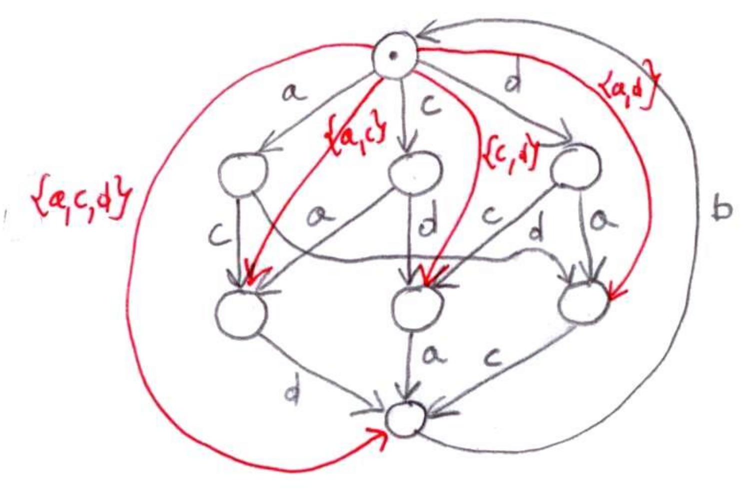


$N_{\|S1}$



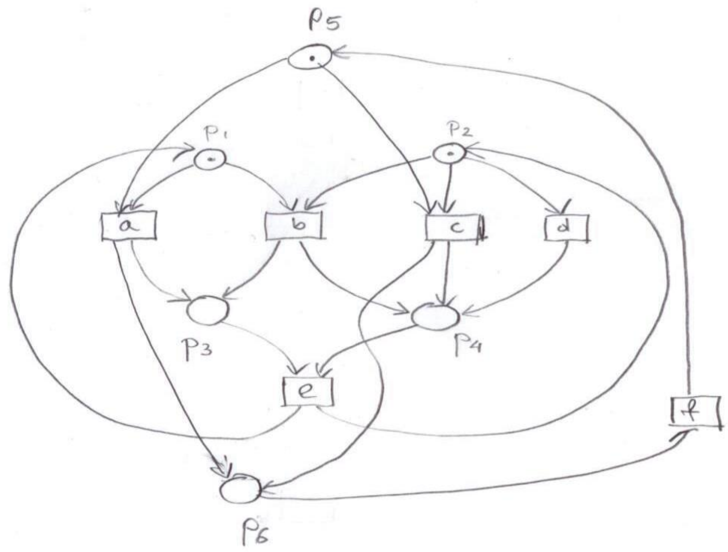
N_{S2}

The net N_{S2} is actually a Labelled Transition System in disguise (see LN3 page 2) and, treated as a Labelled Transition System, it is equal to $LTS_1(S2)$. The labelled transition system LTS(||S1) is nothing but the Reachability Graph of $N_{\|S1}$ (without step traces, see also Question 11 and LN3 page 18). When steps (simultaneous executions) are allowed the Reachability Graph for $N_{\|S1}$ looks as follows:

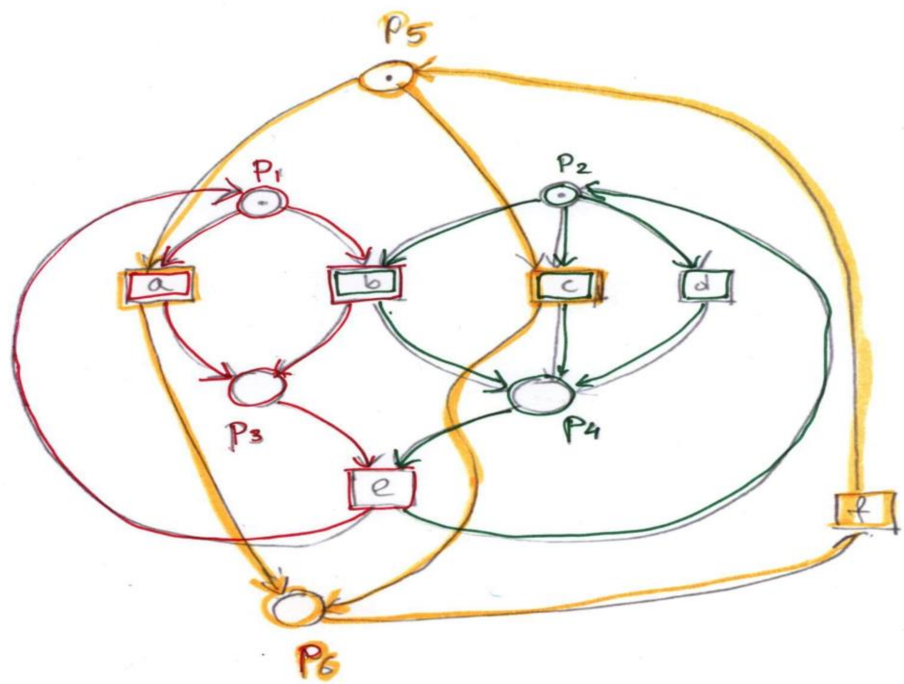


Maximum point does not require such a detailed solutions, however a bonus of up to 10pts will be given for some discussion of the issues mentioned above.

7.[10] Consider the Petri net below:



Model this net as a composition of *FSP* processes.
Solution:
It can easily be decomposed into 3 labelled Transition Systems, Red, Green and Yellow as shown below.



Hence we have:

RED = P1

P1 = (a → P3 | b → P3)
P3 = (e → P1)

GREEN = P2
P2 = (b → P4 | c → P4 | d → P4)
P4 = (e → P2)

YELLOW = P5
P5 = (a → P6 | c → P6)
P6 = (f → P5)

NET = RED || GREEN || YELLOW

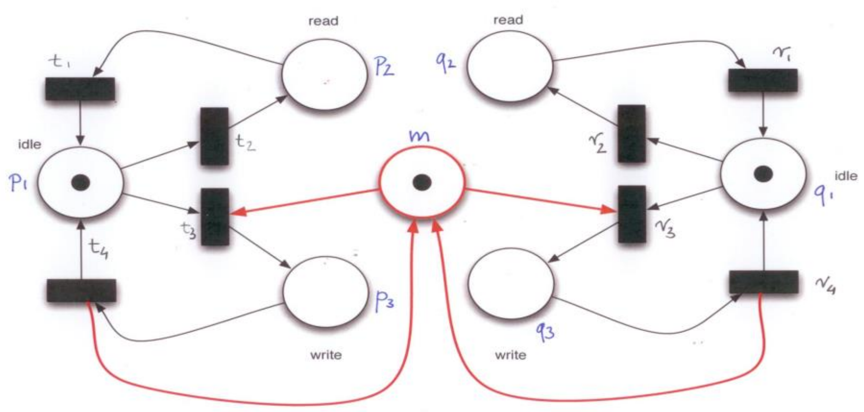
8.[10] Model the system from page 10 of Lecture Notes 3 as a composition of *FSP* processes. In this case, the entities that are represented by places in the Petri Nets model, must be represented by actions/transitions in *FSP* model.

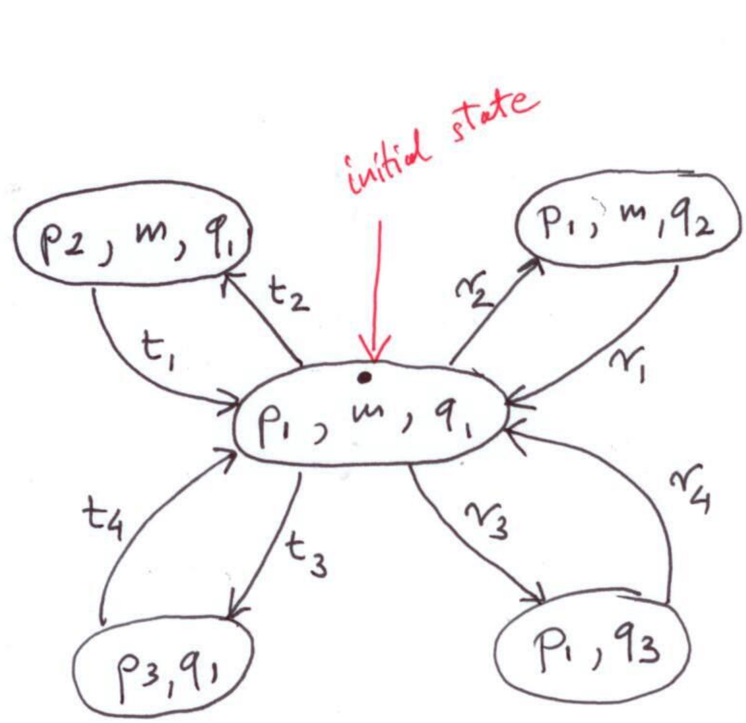
Solution: (a possible one, bonus for using labelling)

```
COMP1 = (idle1 -> (read1 -> COMP1 | write1 -> COMP1))
COMP2 = (idle2 -> (read2 -> COMP2 | write2 -> COMP2))
MUT     = (write1 -> MUT | write2 -> MUT)
||TWPCOMP = COMP1||COMP2||MUT
```

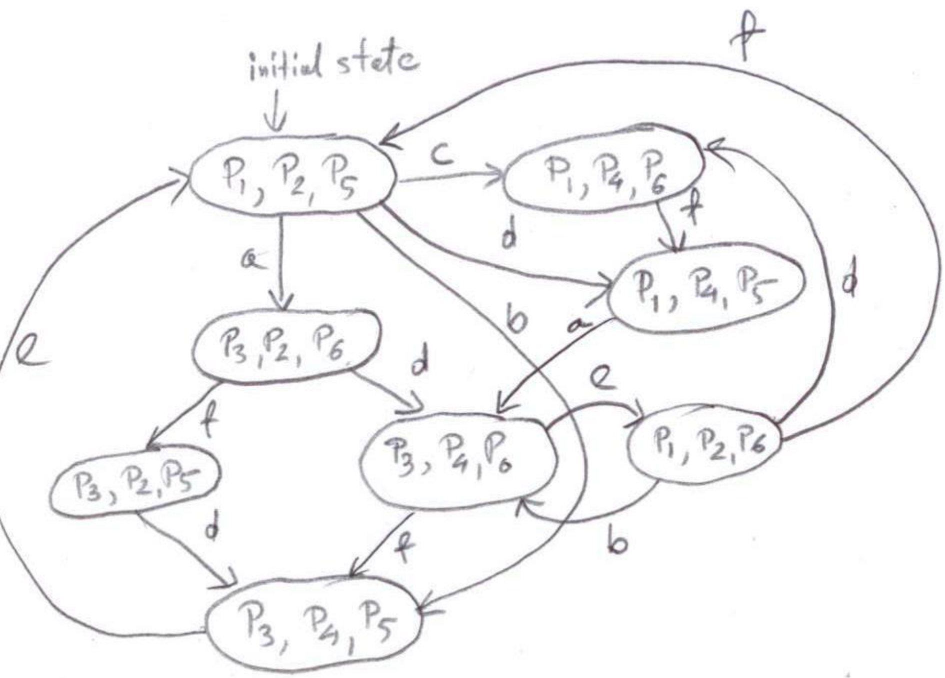
9.[10] Construct *reachability graph* (defined on page 18 of Lecture Notes 3) for the Petri net from Question 8.

Solution: You must add some names for transitions and the mutex place.

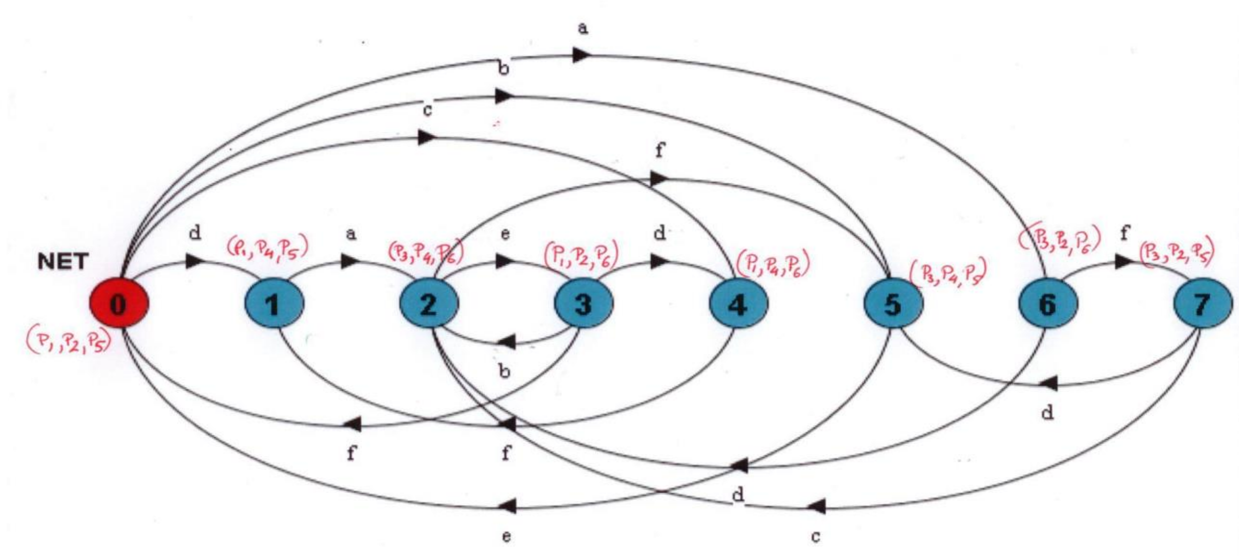




The solution below is for Petri Net from question 7. I leave it as an example.



A solution derived from the solution to Question 7 using LTSA and appropriate labelling of states.



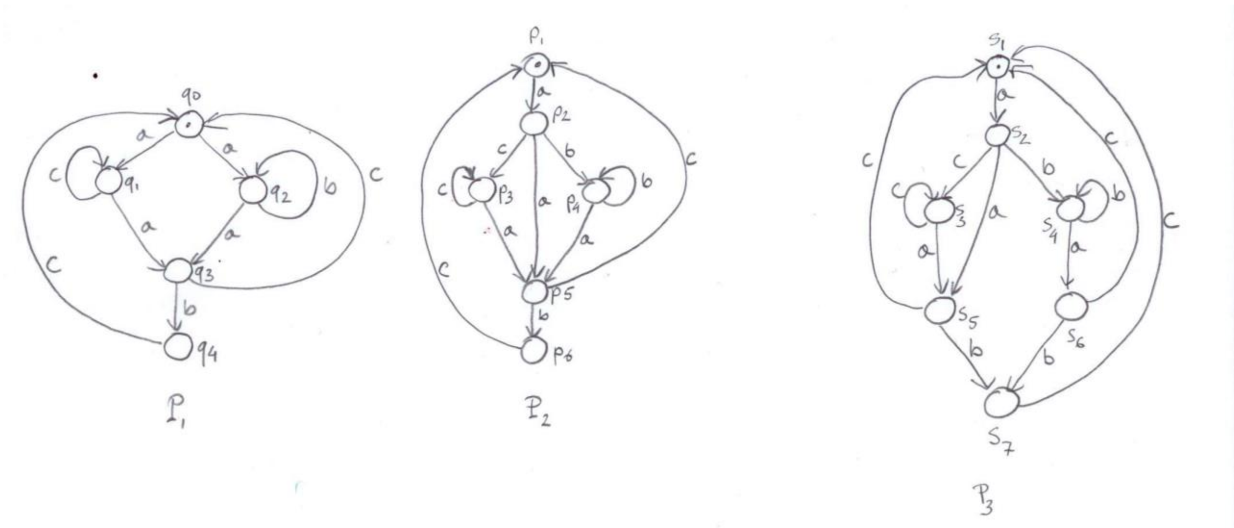
10.[28] Consider three Labelled Transition Systems (Finite State Machines, Finite Automata) given below: P_1, P_2 and P_3. Tokens represent initial states. Show that:

a.[8] $P_2 \approx P_3$, i.e. P_2 and P_3 are *bisimilar*,

b.[6] $P_1 \not\approx P_2$, i.e. P_1 and P_2 are *not bisimilar*,

c.[6] $P_1 \not\approx P_3$, i.e. P_1 and P_3 are *not bisimilar*,

d.[8] Traces(P_1) = Traces(P_2) = Traces(P_3) = Pref(*give a proper regular expression*).



Solutions:

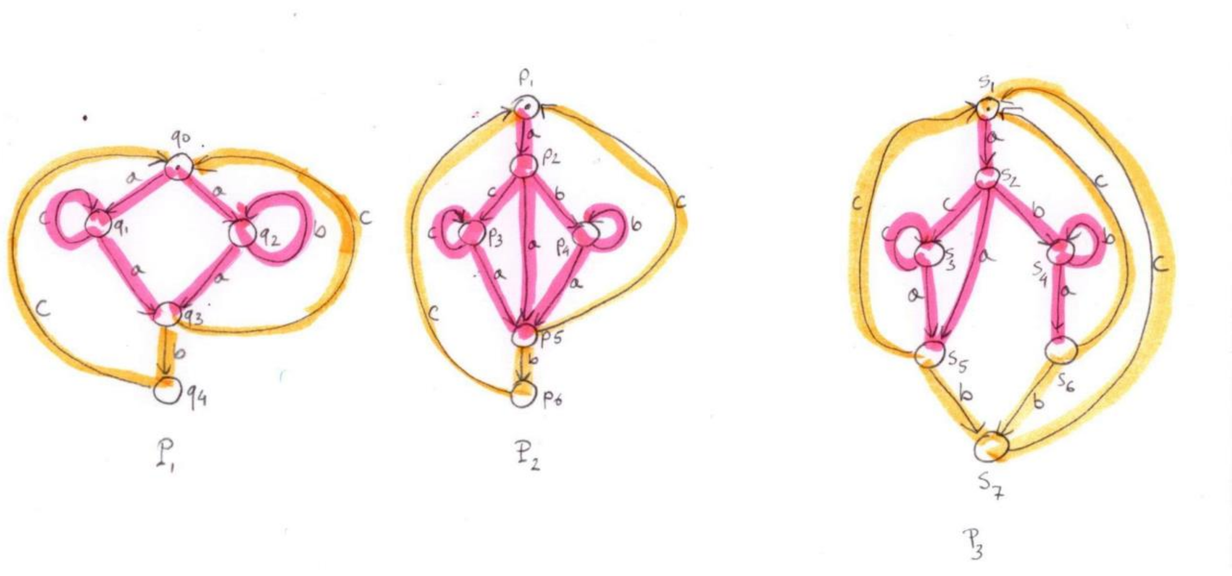
a. p_1 and s_1 are bisimilar as in both cases the action a is allowed.
p_2 and s_2 are bisimilar as they both allow a, b and c.
p_3 and s_3 allow a and c, so they are bisimilar.
P_4 and s_4 are bisimilar as they both allow a and b.
p_5 and s_5 are bisimilar as they both allow b and c.
p_5 and s_6 are bisimilar as they both allow b and c; and
p_6 and s_7 are also bisimilar as they also allow only c.
We have exhausted all cases, so P_2 and P_3 are bisimilar, i.e. $P_2 \approx P_3$.

b. After trace a the labeled transition system P_1 is either in the state q_1 or the state q_2, while P_2 is in the state p_2. In the state p_2 the actions a, b and c are allowed, in the state q_1 the actions a and c are allowed, while in the state q_2 the actions a and b are allowed. Hence both pairs (q_1,p_2) and (q_2,p_2) are *not* bisimilar, i.e. $P_1 \not\approx P_2$.

c. After trace a the labeled transition system P_1 is either in the state q_1 or the state q_2, while P_3 is in the state s_2. In the state s_2 the actions a, b and c are allowed, in the state q_1 the actions a and c are allowed, while in the state q_2 the actions a and b are allowed. Hence both pairs (q_1,s_2) and (q_2,s_2) are *not* bisimilar, i.e. $P_1 \not\approx P_3$.

d. Traces(P_1) = Traces(P_2) = Traces(P_3) = $\mathrm{Pref}(\ ((ac^*a \cup ab^*a)(bc \cup c))^*\ )$

Note that in each case any sequence (trace) that leads from the initial state to the initial state (i.e. from q_0 to q_0 in P_1, p_1 to p_1 in P_2, and s_1 to s_1 in P_3) is a concatenation of RED path and YELLOW path.



In other words: Traces(P_1) = Traces(P_2) = Traces(P_3) = Pref((RED YELLOW)*).
Note that in each labelled transition system (finite automaton)

RED = ab*a ∪ ac*a, and YELLOW = bc ∪ b.

Hence RED YELLOW = (ab*a ∪ ac*a)(bc ∪ b), and

(RED YELLOW)* = ((ab*a∪ ac*a)(bc ∪ b))*.

This means:

Traces(P_1) = Traces(P_2) = Traces(P_3) = Pref(((ac*a ∪ ab*a)(bc ∪ c))*)