COMPSCI 2SD3

ASSIGNMENT 3

PRAKHAR SAXENA

1a)

```
const X = 10
range T = 0..X
CONTROLSYSTEM = CONTROLSYSTEM[0],
CONTROLSYSTEM[i:T] = (when (i>0) exit -> CONTROLSYSTEM[i-1]
                    | when (i<10) enter -> CONTROLSYSTEM[i+1]
                    | when (i==10 || i>0) continue -> CONTROLSYSTEM[i]
                    ).
```

1b)

```
const X = 10
range T = 0..X
property CAPACITY = CAPACITY[0],
CAPACITY[i:t]= (enter -> CAPACITY[i+1]
```

2a)

Let x=m+n+k, then:

R = reading

W = writing

WR = waiting to read

WW = waiting to write

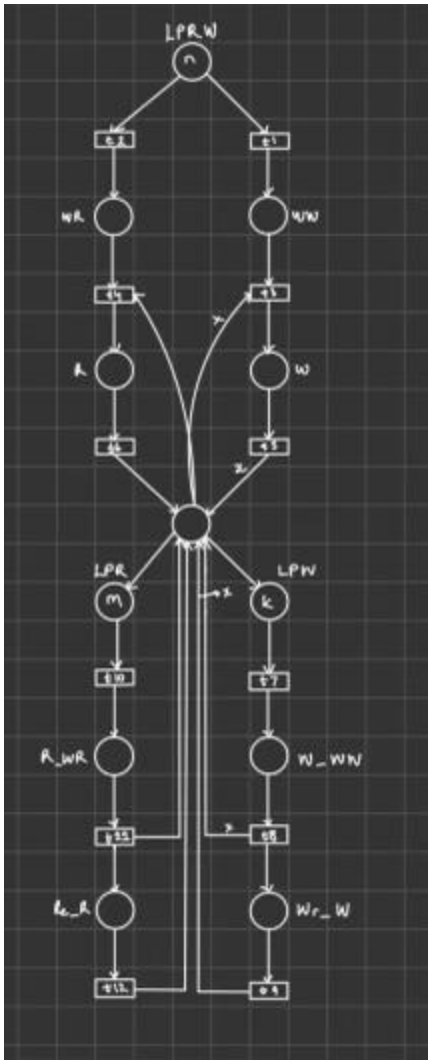
R_WR = readers wait to read

W_WW = writers wait to write

Wr_W = writing for writers

Re_R = reading for readers

S = Synchronization



2b)

There are a few invariants:

Invariant 1:
m(LPWR) + m(WR) + m(WW) + m(W) + m(R) = n

Invariant 2:
m(LPW) + m(Wr_W) + m(W_WW) = k

Invariant 3:
m(LPR) + m(Re_R) + m(R_WR) = m

Invariant 4:
m(LPW) + m(LPR) + m(LPWR) + m(WW) + m(WR) + m(R_WR) + m(W_WW) + m(Wr_W) + m(Re_R) + m(W) + m(R) = x

Invariant 5:
m(Re_R) + x˙m(Wr_W) + m(R) + x˙m(W) + m(s) = s

Proof using invariants:
t_1, t_2, t_7, t_{10}, t_9, t_5, t_6 or t_{12} has concession if:
m(LPW) + m(LPWR) + m(LPR) + m(Wr_W) + m(Re_R) + m(W) + m(R) > 0

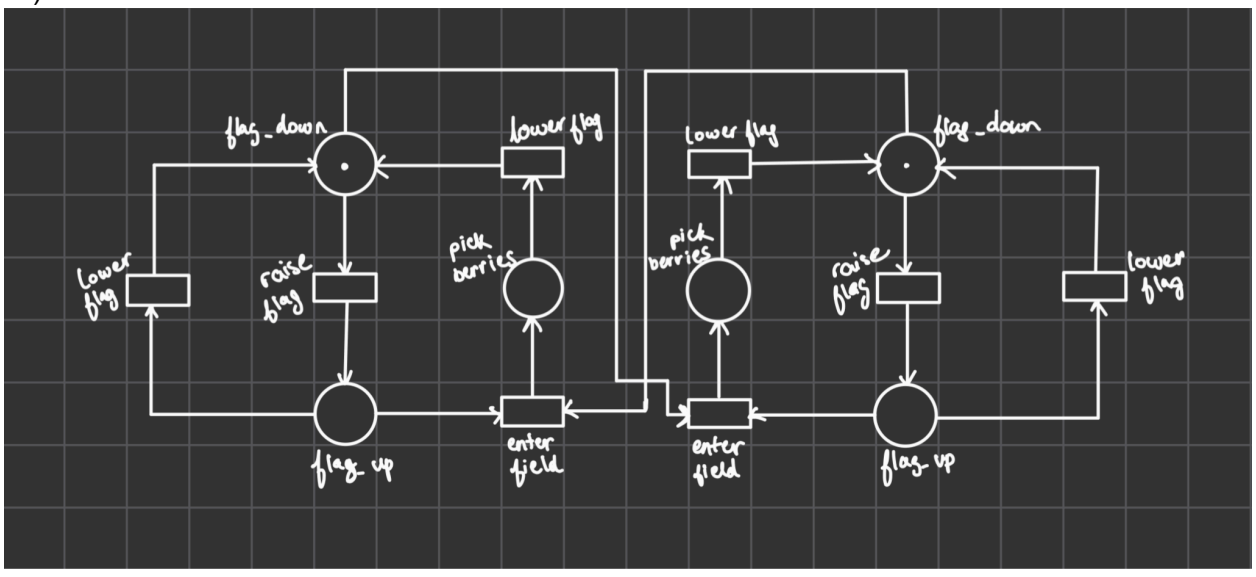
t_3, t_4, t_8, t_{11} has concession if:
m(LPW) + m(LPWR) + m(LPR) + m(W) + m(R) + m(Wr_W) + m(Re_R) = 0

3)
```
Constant C = 1
Range Consumers = 0..A

Constant P =1
Range Producers = 0..B

Constant MP = C+1
Range MixProcesses = 0..C

Consumer (X=0) = (rvcd[X] -> read[X] -> Consumers)
|| Consumers = (forall [i:0..A] c[i] : Consumer(i))

Producer = (type[m:Messages] -> send[Messages]).
||Producers = (forall [i:0..B] p[i] : Producer).

set Q = {[MixProcesses], [MixProcesses][MixProcesses],
[MixProcesses][MixProcesses][MixProcesses]]}

Buffer = (enqueue[i:MixProcesses] -> Buffer[i]),
Buffer[i:MixProcesses] = (enqueue[j:MixProcesses]-> Buffer[i][j]
                          | dequeue[i] -> Buffer),
Buffer[i:MixProcesses][k:Queue] = (enqueue[l:MixProcesses] -> Buffer[i][k][l]
                          | dequeue[i] -> Buffer),
```

4a)
```
const F = 0
const T = 1

range B = F..T
set Boolean = { isT, isF, [T], [F] }

Permit = Bool[False],
Bool[i:B] = ( i = Bool[i]
            | isT -> Bool[T]
            | isF -> Bool[F]
            ).

||Flag = (neighbor1_flag : Permit || neighbor2_flag : Permit).
```

```
Neighbor1 = (neighbor1_flag.isT -> Flag_Up),
Flag_Up = (neighbor2_flag[x:T] ->
                if(x) then
                        (neighbor1_flag.isF -> Neighbor1)
                else
                        (enter -> exit -> neighbor1_flag -> Neighbor1)
            ) + { {neighbor1_flag, neighbor2_flag}.Boolean }

Neighbor2 = (neighbor2_flag.isT -> Flag_Up),
Flag_Up = (neighbor1_flag[x:T] ->
                if(x) then
                        (neighbor2_flag.isF -> Neighbor2)
                else
                        (enter -> exit -> neighbor2_flag -> Neighbor2)
            ) + { {neighbor1_flag, neighbor2_flag}.Boolean }

property Safety = (Neighbor1.enter -> Neighbor1.exit -> Safety | Neighbor2.enter ->
Neighbor2.exit -> Safety ).

|| Berry_Fields = ( Neighbor1 || Neighbor2 || {Neighbor1, Neighbor2} :: Flag || Safety ).

progress X = {Neighbor1.enter}
progress Y = {Neighbor2.enter}

|| Greedy_Neighbor = {{Neighbor1, Neighbor2}.{neighbor1_flag,neighbor2_flag}.isT} >>
Berry_Fields}.
```

4b)



5)
Using variable as S_1 and S_2 and their respective max value we design a Simplified Multidimensional Semaphores.

```
Semaphore (X=InitiatedValue) = Semaphores(X)
Semaphores [e:Int] = (when (e>0) down -> Semaphores[e-1]
                        | when (e<=Max) up -> Semaphore [e+1]
                        ).
SemaphoresS1S2 (InitiatedValue1 = 3, InitiatedValue2 = 3) = (S1 : Semaphore(3) ||  S2 :
Semaphore(3))
/{S1.S2.up/S1.up, S1.S2.up/S2.up, S1.S2.down/S1.down, S1.S2.down/ S1.down}.
```

6a)

```
const N = 3
const M = 2

range A = 1..2
Cust = (pay[x:A] -> gas[y:A] -> if (y==a) then Cust else ERROR).

range B = 1..N
range C = 1..M

Cashier = (customer[x:B] .pay[y:A] -> start [C] [x] [y] -> Cashier).

Pump = (start[x:B][y:A] -> gas[x][y] -> Pump).

DeliverGas = (gas[C][x:b][y:A] -> customer[x] .gas[y] -> DeliverGas).

 ||FILLINGSTATION = (Cashier || pump[1..M] : PUMP || DeliverGas)

                    /pump[x:1..M] .gas/gas[x],

                    {pump[x:1..M] .start/start[x]}.

 ||GASSTATION = (customer[1..N]:Cust ||FILLINGSTATION).
```

6b)

```
const X = 2
range Y = 1..X
property Fifo = (customer[x:Y].pay[A] -> Payment[x])
property Payment[x:Y] = (customer[x] .gas[A] -> Fifo
                          |customer[i:Y] .pay[A] -> Payment[x][i]),
Paid[x:Y][i:Y]= (customer[x].gas[A] -> Payment[i]).
||Check_Fifo = (GASSTATION || Fifo).
```

7a)
i)
$\neg p \Rightarrow r$
Rewriting : $\neg(\neg p) \vee r \equiv p \vee r$
Given, $L(s_0) = \{r\}$ hence, $M, s_0 \models \varphi$
Given, $L(s_2) = \{p,q\}$ hence, $M, s_2 \models \varphi$

ii)
Given, $r \in L(s_0)$ and $L(s_1)$.
Since there are countless paths for ex. $(s_0 \rightarrow s_1 \rightarrow s_1 \rightarrow \ldots..)$ we have $M, s_0 \models EG\ r$ hence, $M, s_0 \not\models \neg EG\ r$
$M, s_s \models \neg EG\ r$ holds as $r \notin L(s_2) = \{p,q\}$.

iii)
From $t \notin L(s_0)$ we get $M, s_0 \not\models E(t\ U\ q)$
From $t \notin L(s_2)$ we get $M, s_2 \not\models E(t\ U\ q)$

vi)
Since we can have countless paths for ex. $(s_0 \rightarrow s_2 \rightarrow \ldots..)$ and because of $q \notin L(s_2)$ we get, $M, s_0 \models F\ q$.
From $q \notin L(s_2)$ we get $s_0 \models F\ q$.

7b)
CTL: $AG(Fp \wedge (AG(p \Rightarrow AFs)) \wedge (AG(p \Rightarrow AFt))$)
LTL: $G(\ Fp \wedge (p \Rightarrow Fs) \wedge (p \Rightarrow Ft))$

7c)
CTL: $AG(Fq \wedge Fr) \wedge AG(q \Rightarrow A(\neg p\ U\ r))$
LTL : $G(\ Fq \wedge Fr \wedge (q \Rightarrow (\neg p\ U\ r) \wedge (q \Rightarrow (Ft\ U\ r)))$

7d)
CTL : $s \models AG(AF\ \Phi)$
LTL : $s \models G(F\ \Phi)$

8)
After taking inspiration from LN15 on mutual exclusion and the naming convention of reader and writer problem:

(STR1, STR2,STR3,STW1,STW2, STW3,turn)

where: $STR1 \in \{LR_1, R\_RR_1, R_1\}$, $STR2 \in \{LR_2, R\_RR_2, R_2\}$, $STR3 \in \{LR_3, R\_RR_3, R_3\}$- readers status;
$STW1 \in \{LW_1, W\_RW_1, W_1\}$, $STW2 \in \{LW_2, W\_RW_2, W_2\}$, $STW3 \in \{LW_3, W\_RW_3, W_3\}$ - writer status;

turn $\in$ {turn=W_1, turn=W_2, turn=W_3, turn=R},

Using,

LR_ii - local processing of reader i,i=1,2,3

LWi - local processing of writer i, i=1,2,3

R_RRi - reader i requests reading, i=1,2,3
W_RWi- writer i requests writing, i=1,2,3

Ri - reader i is reading, i=1,2,3

Wi - writer i is writing, i=1,2,3

However, some atomic predicate combinations are not allowed:

$STW1 = W_1 \Rightarrow STR1 \neq R_1 \wedge STR2 \neq R_2 \wedge STR3 \neq R_3 \wedge STW2 \neq W_2 \wedge STW3 \neq W$

Safety in LTL: $G\ (w_1 \Rightarrow \neg\ (w_2 \vee w_3 \vee r_1 \vee r_2 \vee r_3\ ))$

Liveness in LTL: $G(\ tr_1 \Rightarrow F\ r_1$