2SD3 Assignment 3

Rochan Muralitharan – muralr3

1.

We will use these invariants to prove that this Coloured Petri Net is deadlock-free:

[inv1] $m(p_1) + m(p_2) + m(p_3) + m(p_4) + m(p_5) = ph1 + ph2 + ph3 + ph4 + ph5$

[inv2] $|m(p_2)| + |m(p_7)| = 4$

[inv3] $LF(m(p_4)) + RF(m(p_4)) + m(p_6) = f1 + f2 + f3 + f4 + f5$

Now we will consider these two cases:

a. $m(p_4) + m(p_5) \neq 0$.
   Then either return_left_fork or return_right_fork and exit_dining_room can be fired

b. $m(p_4) + m(p_5) = 0$.
   Then from invariant [inv3] we will have:
   $LF(m(p_3)) + m(p_6) = f1 + f2 + f3 + f4 + f5$
   and from invariant [inv1]:
   $m(p_1) + m(p_2) + m(p_3) = ph1 + ph2 + ph3 + ph4 + ph5$

From the definition of LF(x) and the definition of RF(x), we know that we have x = ph1, ph2, ph3, ph4, ph5.

Thus if $m(p_3) \neq 0$ then the transition take_right_fork can be fired.
Also if $m(p_2) \neq 0$ then take_left_fork can be fired.

If $m(p_1) \neq ph1 + ph2 + ph3 + ph4 + ph5$, then either $m(p_3) \neq 0$ or $m(p_2) \neq 0$.

If $m(p_1) = ph1 + ph2 + ph3 + ph4 + ph5$ then $m(p_2) = 0$, and from invariant [inv2] $|m(p_7)| = 4$, so enter_dining_room can be fired.

2. a)

```
const N = 3 // customers
const M = 2 //pumps

range C = 1..N
range P = 1..M
range A = 1..2

CUSTOMER = (prepay[a:A] -> gas[x:A] ->
                if (x==a) then CUSTOMER else ERROR).

CASHIER = (customer[c:C].prepay[x:A] -> start[P][c][x] -> CASHIER).

PUMP = (start[c:C][x:A] -> customer[c].gas[x] -> DELIVER).

DELIVER = (gas[P][c:C][x:A] -> customer[c].gas[x] -> DELIVER).

||STATION = (CASHIER || pump[P]:PUMP || DELIVER)
            /{pump[i:P].start/start[i],
              pump[i:P].gas/gas[i]}.

||GASSTATION = (customer[C] : CUSTOMER || STATION).
```

b)

```
range T = 1..2

property
    FIFO = (customer[i:T].prepay[A] -> PAID[i]),
    PAID[i:T] = (customer[i].gas[A] -> FIFO | customer[j:T].prepay[A] -> PAID[i][j]),
    PAID[i:T][j:T] = (customer[i].gas[A] -> PAID[j]).

||CHECK_FIFO = (GASSTATION || FIFO).
```

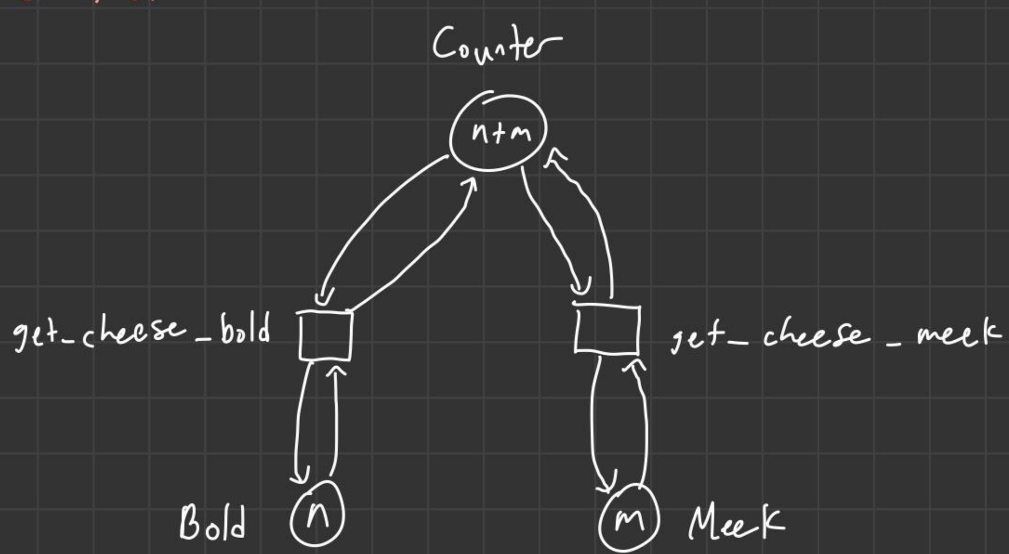
c) In Java File Gas_Station_A3_Q2.java

3. a)

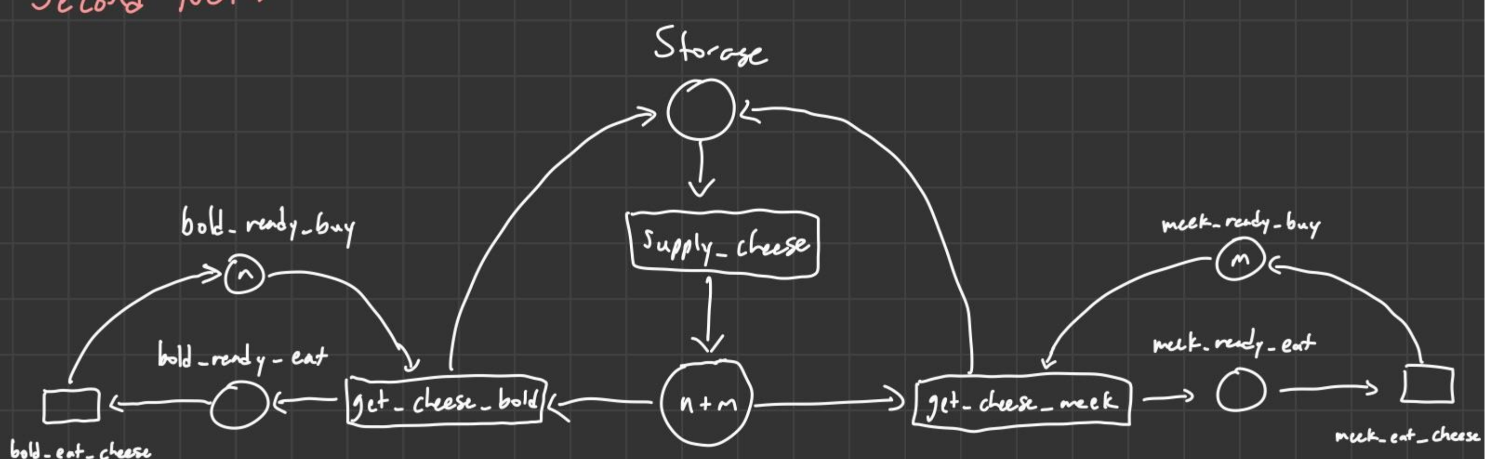
```
set Bold = {bold[1..2]}
set Meek = {meek[1..2]}
set Customers = {Bold, Meek}

CUSTOMER = (getcheese -> CUSTOMER).

COUNTER = (getcheese -> COUNTER).

||CHEESECOUNTER = (Customers: CUSTOMER || Customers:: COUNTER).
```

3. b) We can model this system with 2 Place/Transition Nets

First Net:



Second Net:



3.c)

```
set Bold = {bold[1..2]}
set Meek = {meek[1..2]}
set Customers = {Bold, Meek}

CUSTOMER = (getcheese -> CUSTOMER).

COUNTER = (getcheese -> COUNTER).

||CHEESECOUNTER = (Customers: CUSTOMER || Customers:: COUNTER)>>{Meek.getcheese}.

progress BOLD = {Bold.getcheese}

progress MEEK = {Meek.getcheese}
```

Here we can see that Meek.getcheese will clearly get starved, since Bold.getcheese will always be executed. Bold will always be given favour when there is a choice between meek and bold.

4.

```
set Bold = {bold[1..2]}
set Meek = {meek[1..2]}
set Customers = {Bold, Meek}
const MAX = 4
range T = 1..MAX

CUSTOMER = (ticket[t:T] -> getcheese[t] -> CUSTOMER).

TICKET = TICKET[1],
TICKET[t:T] = (ticket[t] -> TICKET[t%MAX+1]).

COUNTER = COUNTER[1],
COUNTER[t:T] = (ticket[t] -> COUNTER[t%MAX+1]).

||CHEESECOUNTER = (Customers: CUSTOMER || Customers:: TICKET ||
                   Customers:: COUNTER) >> {Meek.getcheese[T]}.

progress BOLD = {Bold.getcheese[T]}

progress MEEK = {Meek.getcheese[T]}
```

5. In Java File A3_Q5.java

6.

```
const N = 3
set M = {msg}
set S = {[M], [M] [M]}

PORT = (send[x:M] -> PORT[x]),
PORT [y:M] = (send[x:M] -> PORT[x][y] | receive[y] -> PORT),
PORT[z:S][y:M] = (send[x:M] -> PORT[x][z][y] | receive[y] -> PORT[z]).

PRODUCER = (empty.receive.token -> dest.send.msg -> PRODUCER).

CONSUMER = SENDBUF[N],
SENDBUF[i:1..N] = (empty.send.token -> if (i == 1) then CONTINUE else SENDBUF[i-1]),
CONTINUE = (dest.receive.msg -> empty.send.token -> CONTINUE).

||PROCON = (PRODUCER || CONSUMER || empty:PORT || dest: PORT)
            /{empty.[i:{send,receive}].token/empty[i].msg}.
```

7. a)

(i) $\varphi = (\neg p \Rightarrow r)$: φ is equivalent to $(\neg(\neg p) \vee r) \equiv p \vee r$. We have $L(s_0) = \{r\}$ so $M, s_0 \vDash \varphi$. We also have $L(s_2) = \{p,q\}$ so $M, s_0 \vDash \varphi$

(ii) $\varphi = \neg$ EG r → This statement translates to: There does not exist at least one path from all future states leading to r. We have $r \in L(s_0)$ and $r \in L(s_1)$ as $L(s_0) = \{r\}$ and $L(s_1) = \{p,t,r\}$. There is an infinite path $s_0 \rightarrow s_1 \rightarrow s_1 \rightarrow s_1 \rightarrow \dots$, so $M, s_0 \vDash$ EG r and thus we can infer that $M, s_0 \nvDash \varphi$. Also, $r \notin L(s_2)$ as $L(s_2) = \{p, q\}$ and thus $M, s_2 \vDash \varphi$ as the future includes the present.

(iii) $\varphi = E(t\ U\ q)$ → This statement translates to: There exists at least one path in where t will occur until q. As $t \notin L(s_0)$ and $t \notin L(s_2)$, we have $M, s_0 \nvDash \varphi$ and $M, s_2 \nvDash \varphi$.

(iv) $\varphi = F\ q$ → This statement translates to: Some future state leads to q. As $q \in L(s_2)$ since $L(s_2) = \{p,q\}$, and there are infinite paths $s_0 \rightarrow s_2 \rightarrow s_0 \rightarrow s_2 \rightarrow \dots$, we have $M, s_0 \vDash \varphi$. Also trivially, $M, s_2 \vDash \varphi$ as $q \in L(s_2)$ and the futures includes the present.

For the following questions we will assume the following:

“p precedes q” means that p must happen before q, and not at the same time
“p is followed by q” means that q must happen before p, and not at the same time
“p is between q and r” means that p does occur at the same time as q or r

b) “Event p precedes s and t on all computational paths”
Negation: “There exists a path where p does not precede s or does not precede t”

LTL: $G(F\ p \wedge (p \Rightarrow F\ s) \wedge (p \Rightarrow F\ t))$
CTL: $AG(AF\ p \wedge AG(p \Rightarrow AF\ s) \wedge AG(p \Rightarrow AF\ t))$

c) "Between the events q and r, p is never true but t is always true"
LTL: $G(F\ p \wedge F\ r \wedge (q \Rightarrow (\neg\ p\ U\ r) \wedge (q \Rightarrow (F\ t\ U\ r)))$
CTL: $AG(AF\ q \wedge AF\ r) \wedge AG\ (q \Rightarrow A(\neg\ p\ U\ r))$

d) "ϕ is true infinitely often along every path starting at s"
LTL: $s \vDash G(F\ \phi)$
CTL: $s \vDash AG(AF\ \phi)$

e) "Whenever p is followed by q(after some finite amount of steps), then the system enters an 'interval' in which no r occurs until t"
LTL: $G(p \Rightarrow XG(\neg\ q \vee \neg\ r\ U\ t))$
CTL: $AG(p \Rightarrow AX\ AG(\neg\ q \vee A\ (\neg\ r\ U\ t)))$

f) "Between the events q and r, p is never true"
LTL: $G(F\ q \wedge F\ r \wedge (q \Rightarrow (\neg\ p\ U\ r)))$
CTL: $AG(AF\ q \wedge AF\ r) \wedge AG(q \Rightarrow A(\neg\ p\ U\ r))$

8. We will assume the following atomic predicates that characterize properties of processes:

lpr_i = local processing of reader i, i=1,2
lpw_i = local processing of writer i, i=1,2
tr_i = reader i, i=1,2, requests reading
tw_i = writer i, i=1,2, requests writing
r_i = reader i i=1,2, is reading
w_i = writer i, i=1,2, is writing

To avoid any problems that might occur if we do not consider mutual exclusion, we will introduce some additional boolean variables(or atomic predicates):

turn = w1 (indicates the world where writer 1 will write)
turn = w2 (indicates the world where writer 2 will write)
turn = r (indicates the world where one or both readers will read)

Now the states can be identified by the atomic predicates of the form:

(sr1, sr2, sw1, sw2, turn)

Where:

sr1 ∈ {lpr_1, tr_1, r_1} - status of reader 1

$sr2 \in \{lpr_2, tr_2, r_2\}$ – status of reader 2
$sw1 \in \{lpw_1, tw_1, w_1\}$ - status of writer 1
$sw2 \in \{lpw_2, tw_2, w_2\}$ – status of writer 2
$turn \in \{turn = w1, turn = w2, turn = r\}$ – status of turns

Life of a reader follows the simple cycle:

$(lpr_1, *, *, *, *) \longrightarrow (tr_1, *, *, *, *) \longrightarrow (r_1, *, *, *, *) \longrightarrow$ back to the beginning

Life of a writer follows a similar cycle:

$(*, *, lpw_1, *, *) \longrightarrow (*, *, tw_1, *, *) \longrightarrow (*, *, w_1, *, *) \longrightarrow$ back to the beginning

However not all combinations of atomic predicates are allowed, for example:

$$sw1 = w_1 \Rightarrow sr1 \neq r_1 \wedge sr2 \neq r_2 \wedge sw2 \neq w_2$$

OR

$$sr1 = r_1 \Rightarrow sw1 \neq w_1 \wedge sw2 \neq w_2$$

Now we can establish safety and liveness properties in LTL and CTL:

Safety

LTL: $G(w_1 \Rightarrow \neg(w_2 \vee r_1 \vee r_2))$
CTL: $AG(w_1 \Rightarrow \neg(w_2 \vee r_1 \vee r_2))$

Liveness

LTL: $G(tr_1 \Rightarrow F\ r_1)$
CTL: $AG(tr_1 \Rightarrow AF\ r_1)$