CS 3BB4. Sample solutions to the assignment 3.

Total of this assignment is 138 pts. Each assignment is worth 10% of total.

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 during class, office hours, or just slip under the door to my office. The deadline for a complaint is 2 weeks after the assignment is marked and returned

1.[10] A lift has a maximum property of ten people. In the model of the lift control system, passengers entering a lift are signalled by an `enter` action and passengers leaving the lift are signalled by an `exit` action.

a.[5] Provide a model of the lift in FSP.

b.[5] Specify a safety property in FSP which when composed with the lift check that the system never allows the lift it controls to have more than ten occupants.

Solution

a.
```
const N = 10

LIFT = LIFT[0],
LIFT[i:0..10] = (when(i<10) enter -> LIFT[i+1]
                |when(i>0) exit  -> LIFT[i-1]
                |when(i==10)go   -> LIFT[i]  //lift is full
                |when(i>0) go  -> LIFT[i]   //lift no longer waits
                ).
```

b.
```
const N = 10

property LIFTCAPACITY = LIFTC[0],
LIFTC[i:0..10] = (enter -> LIFTC[i+1]
                 |when(i>0) exit  -> LIFTC[i-1]
                 |when(i==0)exit  -> LIFTC[0]
                 ).
```

2.[12] Specify a safety property for th car park problem of Chapter 5 of the textbook and Lecture Notes 7, which asserts that the car park does not overflow. Specify a progress property which asserts that cars eventually enter the car park. If car departure is lower priority (as defined by the textbook) than car arrival, does starvation occur?

Solution

```
CARPARKCONTROL(N=4) = SPACES[N],
SPACES[i:0..N] = (when(i>0) arrive->SPACES[i-1]
            |when(i<N) depart->SPACES[i+1]
            ).

ARRIVALS   = (arrive->ARRIVALS).
DEPARTURES = (depart->DEPARTURES).

||CARPARK = (ARRIVALS||CARPARKCONTROL(4)||DEPARTURES).

property OVERFLOW(N=4) = OVERFLOW[0],
OVERFLOW[i:0..N] = (arrive -> OVERFLOW[i+1]
                  |depart -> OVERFLOW[i-1]
                  ).

||CHECK_CARPARK = (OVERFLOW(4) || CARPARK).

/* try safety check with OVERFLOW(3) */

progress ENTER = {arrive}

||LIVE_CARPARK = CARPARK >>{depart}.
```

3.[20] The net model considered in Lecture Notes 12 assumed that each process can either read or write. Suppose that we have *three* kind of processes, *readers* that can only read, *writers* that can only write, and *rw-processes*, that can both read and write. Assume we have *m readers, k writers* and *n rw-processes*.

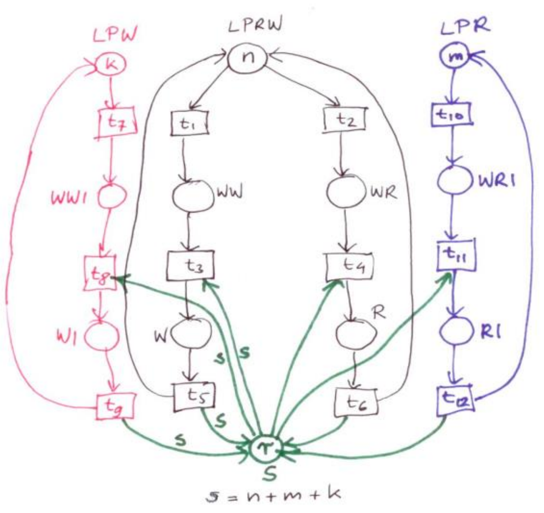
a.[10] Provide a Petri nets solution in the style of Lecture Notes 12 for this version of Readers and Writers problem. The solution should be for an **arbitrary** *n*, *m*, and *k*, **not** for any specific values of *n*, *m* and *k* (as for example *n*=3, *m*=2, *k*=3).

b.[10] Prove that your solution is deadlock-free by mimicking the proof of Proposition from page 27 of Lecture Notes 12.

Solution:

a. We assume s=n+m+k. Black part: rw-processes, red: writers, blue: readers, green: tickets
WW - waiting to write for rw-processes
WR - waiting to read for rw-processes
W - writing for rw-processes
R - reading for rw-processes
WW1 - waiting to write for writers
WR1 - waiting to read for readers
W1 - writing for writers
R - reading for readers
S - tickets (synchronization)

Note that we *cannot* join WW and WW1 into one place and similarly for WR and WR1, W and W1, R and R1.



b.
There are several obvious invariants. Let s=n+m+k.
in1: m(LPWR)+m(WW)+m(WR)+m(W)+m(R)=n
in2: m(LPW)+m(WW1)+m(W1)=k
in3: m(LPR)+m(WR1)+m(R1)=m
in4: m(LPW)+m(LPWR)+m(LPR)+m(WW1)+m(WW)+m(WR)+m(WR1)+
m(W1)+m(W)+m(R)+m(R1)=s
The invariant in4 follows from in1, in2 and in3 but it is very useful in explicit form.
in5: m(R)+sAm(W)+m(R1)+sAm(W1)+m(s)= s

If m(LPW)+m(LPWR)+m(LPR)+m(W1)+m(W)+m(R)+m(R1)>0, then at least one of 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(W1)+m(W)+m(R)+m(R1)=0, from in4 we have:

m(WW1)+m(WW)+m(WR)+m(WR1)=s,

and from in5:

m(WR)+m(WR1)+m(WW)+m(WW1)=s
m(S)=s,

so either t_8 or t_3, or at least one of t_4, t_{11} has concession.

4.[30] A self-service gas station has a number of pumps for delivering gas to customers for their vehicles. Customers are expected to prepay a cashier for their gas. The cashier activates the pump to deliver gas.

a.[10] Provide a model for the gas station with *N* customers and *M* pumps. Include in the model a range for different amounts of payment and that customer is not satisfied (`ERROR`) if incorrect amount of gas is delivered.

b.[10] Specify and check (with *N*=2, *M*=3) a safety property FIFO (First In First Out), which ensures that customers are served in the order in which they pay.

c.[10] Provide a simple Java implementation for the gas station system with *N*=2, *M*=3.

Solutions:

a.

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

range C = 1..N
range P = 1..M
range A = 1..2   //Amount of money or Gas

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] -> gas[c][x]->PUMP).

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

||STATION = (CASHIER || pump[1..M]:PUMP || DELIVER)
            /{pump[i:1..M].start/start[i],
              pump[i:1..M].gas/gas[i]}.

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

This property is expected to be violated.

c. Java solutions are not provided.

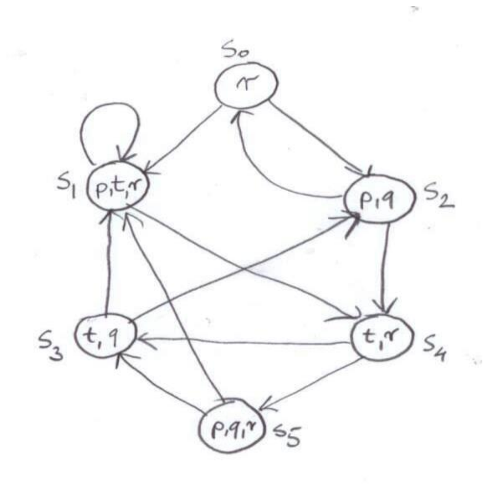
5.[10] Extend the master-slave model of section 9.8 of the textbook and Lecture Notes 16 to cater for N slave processes.

Solution:

```
const N = 3

SLAVE = (start->rotate->join->SLAVE).

MASTER = START[1],
START[i:1..N] = (slave[i].start -> if (i<N) then START[i+1] else
ROTATE),
ROTATE        = (rotate -> JOIN[1]),
JOIN[i:1..N]  = (slave[i].join  -> if (i<N) then JOIN[i+1] else
(rotate->MASTER)).


||MASTER_SLAVE = (MASTER || slave[1..N]:SLAVE).
```

6.[36] This question deals with Model Checking.

(a)[12] Consider the system *M* defined below:



Determine whether *M*, s_0 |= φ and *M*, s_2 |= φ hold and justify your answer, where φ is the LTL or CTL formula:

(i)[3] $\neg p \Rightarrow r$

(ii)[3] $\neg$ EG r

(iii)[3] E(t U q)

(iv)[3] F q

(b)[8] Express in LTL and CTL: 'Event *p* precedes *s* and *t* on all computational paths' (You may find it easier to code the negation of that specification first).

(c)[8] Express in LTL and CTL: 'Between the events *q* and *r*, *p* is never true but *t* is always true'.

(d)[8] Express in CTL: 'Φ is true infinitely often along every paths starting at s'. What about LTL for this statement?

Solutions:

a. (i) $(\neg p \Rightarrow r)$ is equivalent to $(\neg(\neg p) \vee r) \equiv p \vee r$. We have $L(s_0)=\{r\}$ so $M, s_0 \models \varphi$. We have $L(s_2)=\{p,q\}$, so $M, s_0 \not\models \varphi$.

(ii) We have $r \in L(s_0)$ and $r \in L(s_1)$. Moreover there is an infinite path $s_0 \to s_1 \to s_1 \to s_1 \to \ldots$, so $M,s_0 \models$ EG r. Therefore, we infer $M,s_0 \not\models \neg$ EG r.
Since $r \notin L(s_2)=\{p,q\}$, so $M, s_2 \models \neg$ EG r as *future includes present.*

(iii) See LN 15, page 68. Since $t \notin L(s_0)$ and $t \notin L(s_2)$, we have $M,s_0 \not\models$ E(t U q), and $M,s_2 \not\models$ E(t U q).

(vi) We have a path $s_0 \to s_1 \to s_1 \to s_1 \to \ldots$, $q \notin L(s_0)$, and $q \notin L(s_1)$, so $M,s_0 \not\models$ F q.

b. Statement: '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*'.

Ambiguities: Is the case when *p* never happens allowed? We assume it is not (which means 'yes' for negation). Does 'precede' allows *p* and *s* (or *p* and *t*) be in the same state? We assume it is not (which means 'yes' for negation).

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

c. Statement: 'Between the events *q* and *r*, *p* is never true but *t* is always true'
Ambiguities: Is the case when r or q never happens allowed? We assume that it is not.
What exactly "between" means? We assume "between" is "closed interval" so p is false in the state that holds q and in the state that holds r.

LTL: $G(F\,q \wedge F\,r \wedge (q \Rightarrow (\neg\, p\ U\ r) \wedge (q \Rightarrow (Ft\ U\ r)))$
CTL: $AG(F\,q \wedge F\,r) \wedge AG(\,q \Rightarrow A(\neg\, p\ U\ r))$

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

7.[20] Consider *Readers-Writers* as described on page 14 of Lecture Notes 12 and analysed in Lecture Notes 12 after page 14. Take the case of three processes and provide a model in LTL or CTL.

You have to provide a state machine that defines the model as figures on pages 30 and 33 of Lecture Notes 15 for *Mutual Exclusion*, appropriate atomic predicates as n_1, n_2, t_1, t_2, c_1, c_2 for Mutual Exclusion, and appropriate safety and liveness properties.

Solution:

Solutions are structurally similar to Mutual Exclusion that was considered in class. Assume the following atomic predicates that characterise properties of processes:

lpr_i - local processing of *reader i*, *i*=1,2,3
lpw_i - local processing of *writer i*, *i*=1,2,3
tr_i - *reader i*, *i*=1,2,3 requests reading,
tw_i - *writer i*, *i*=1,2,3 requests writing,
r_i - *reader i*, *i*=1,2,3 is reading,
w_i - *writer i*, *i*=1,2,3 is writing,

Note that the solution restricted to writers only should be the same as Mutual Exclusion considered in class! Hence to avoid similar problems we have to introduced additional boolean variables (or atomic predicates): turn=w1, turn=w2, turn=w3 and turn=r, to indicate that worlds where writer 1 will write (turn=w1), writer 2 will write (turn=w2), writer 3 will write (turn=w3) or readers (one or both) will read (turn=r).

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

(str1, str2,str3,stw1,stw2, stw3,turn)

where: str1 $\in \{lpr_1, tr_1, r_1\}$, str2 $\in \{lpr_2, tr_2, r_2\}$, str3 $\in \{lpr_3, tr_3, r_3\}$- status of readers;
stw1 $\in \{lpw_1, tw_1, w_1\}$, str2 $\in \{lpw_2, tw_2, w_2\}$, str3 $\in \{lpw_3, tw_3, w_3\}$ - status of writerss;
turn $\in$ {turn=w1, turn=w2, turn=w3, turn=r}, - status of turns.

Life of a reader is a simple cycle:
$(lpr_1, *,*,*,*,*,*) \rightarrow (tr_1, *,*,*,*,*,*) \rightarrow (r_1, *,*,*,*,*,*) \rightarrow$ back to beginning,
similarly for writers: $(*,*,*,lpw_1,*,*,*) \rightarrow (*,*,*,tw_1,*,*,*) \rightarrow (*,*,w_1,*,*,*) \rightarrow$ back to beginning.

Not all combinations of atomic predicates are allowed, for example
$stw1 = w_1 \Rightarrow str1 \neq r_1 \wedge str2 \neq r_2 \wedge str3 \neq r_3 \wedge stw2 \neq w_2 \wedge stw3 \neq w_3$, or
$str1 = r_1 \Rightarrow stw1 \neq w_1 \wedge stw2 \neq w_2 \wedge stw3 \neq w_3$.

Properties are also very similar to these for Mutual Exclusion:

Safety in LTL: $G(w_1 \Rightarrow \neg(w_2 \vee w_3 \vee r_1 \vee r_2 \vee r_3))$, or in CTL: $AG(w_1 \Rightarrow \neg(w_2 \vee w_3 \vee r_1 \vee r_2 \vee r_3))$, etc.
Liveness in LTL: $G(tr_1 \Rightarrow F\, r_1)$, or in CTL: $AG(tr_1 \Rightarrow AF\, r_1)$, etc.

The remaining analysis is also similar to Mutual Exclusion from the Lecture Notes.