CS412 Exercise sheet 6

Structuring

1. Suppose the following machine controls a light switch.

```
MACHINE         Light
SETS            POSITION = {on,off}
VARIABLES       switch
INVARIANT       switch : POSITION
INITIALISATION  switch := off
OPERATIONS
  switchoff =
    PRE switch = on THEN switch := off END;
  switchon =
    PRE switch = off THEN switch := on END;
  oo <-- switchstatus = oo:=switch
END
```

A door control is intended to allow the door to be open only if the light is on.

(a) What is wrong with the following?

```
MACHINE         Door
SEES            Light
SETS            DSTATE = {open,closed}
VARIABLES       door
INVARIANT       door : DSTATE & (door = open => switch = on)
INITIALISATION  door := closed
OPERATIONS
  closedoor =  PRE door = open   THEN door := closed END;
  opendoor = PRE door = closed THEN door := open   END
END
```

(b) Would an ammendment to the Door machine which removed the second conjunct of the invariant solve the problem?

(c) How could the specification be altered to work as required?

(d) Write down and verify the initialisation condition for the Door machine (with invariant as printed above).

2. The machine Fifo models a first-in-first-out queue.

```
MACHINE          Fifo(ELEM,cap)
CONSTRAINTS      cap : NAT1
VARIABLES        contents
INVARIANT        contents : seq(ELEM) & size(contents) <= cap
INITIALISATION   contents := <>
OPERATIONS
  input(ee) =
     PRE  ee : ELEM & size(contents) < cap
     THEN contents := contents <- ee
     END;
  ee <-- output =
     PRE  size(contents) > 0
     THEN ee := first(contents) || contents := tail(contents)
     END
END
```

The Router machine makes use of the Fifo specification.

```
MACHINE          Router
INCLUDES         Fifo(MSG, qmax)
SETS             MSG; DEST; STATUS = {yes,no}
CONSTANTS        qmax
PROPERTIES       qmax : NAT1
VARIABLES        pending, is_pending, nexthop
INVARIANT        nexthop : MSG --> DEST & pending : MSG &
                 is_pending : STATUS
INITIALISATION   nexthop :: MSG --> DEST || pending :: MSG ||
                 is_pending := no
OPERATIONS
  receive(mm:MSG) =
    PRE  mm:MSG
    THEN IF   size(contents) < qmax
         THEN input(mm)
    END
    END;
  retreive =
    IF   size(contents) > 0 & is_pending = no
    THEN pending <-- output || is_pending := yes
    END;
  ndest,msg <-- forward =
    IF   is_pending = yes
    THEN msg := pending || ndest := nexthop(pending) || is_pending := no
    END
END
```

(a) Using the condition given in lectures generate the condition for operation receive and show that it holds.

(b) Suppose that the receive operation did not have the IF statement but merely called input(mm). What problem occurs with this and where would it show up in the proof?

(c) If you have time, look at the proof for the next operation, retreive.

After attempting this you will appreciate the benefits of having a tool to organise the proofs and discharge all the simple obligations! You can try this out in the B Tool and it will autoprove immediately.

3. Although a machine can only be controlled by one machine which includes it, we may want to have different instantiations of a particular machine. For example, we might want to specify a machine which operates two Fifo queues of messages. Or we might want queues of completely different types and capacities. To include different copies of a single machine you can prefix the machine name by a distinguishing tag, eg:

```
INCLUDE   copy1.Fifo(TYPE1,cap1), copy2.Fifo(TYPE1,cap2)
```

The variable names and operations of each machine will be tagged in accordance with this to distinguish between them.

Suppose a system places incoming job requests on a Fifo queue. A further operation removes a request and either discards it (if not an appropriate request for this system) or places it on a second Fifo queue whence it will be retrieved for processing. Write an abstract machine to specify this system.