1. Binary decision diagram for \( f(x,y,z) = (!x+y+!z) \cdot (x+y+z) \cdot (x+y) \)

Solutions:

a) \( [x,y,z] \)

b) \( [y,x,z] \)
c) \([y,x,z]\)

d) With 6 edges, the reduced OBDD with orderings \([z,x,y]\) and \([z,y,x]\) have the minimal number of edges.

2. SMV warm-up with semaphore.
See the SMV paper for what “semaphore.smv” should look like. The added spec is:

\[
\text{AG (proc1.state = entering} \rightarrow \text{AF proc1.state = critical)}
\]

The spec is not satisfied and SMV provides a counterexample where proc1 wants to enter the critical section (state = entering), proc2 goes into a loop where it repeatedly enters the critical section and returns to idle, while proc1 only executes while proc2 is in its critical section.

3. Elevator modeling

**Solutions:**

a) Elevator model:

```plaintext
-- CSE 590 HW4
MODULE main
VAR
  -- cabin state variable
  cabin : 0 .. 3;
  -- direction variable
  dir : (up, down);
  -- array of requests
  request : array 0 .. 3 of boolean;
ASSIGN
  next(cabin) :=
      case
        dir = up & cabin < 3 : cabin + 1; -- up
        dir = down & cabin > 0 : cabin - 1; -- down
        1 : cabin; -- stuck
      esac;
  next(dir) :=
```
case
dir = up & next(cabin) = 3 : down; -- switch to down
dir = down & next(cabin) = 0 : up; -- switch to up
1 : dir; -- continue
esac;
next(request[0]) :=
case
next(cabin) = 0 : 0; -- satisfied
request[0] : 1; -- continue
1 : {0,1}; -- new request?
esac;
next(request[1]) :=
case
next(cabin) = 1 : 0; -- satisfied
request[1] : 1; -- continue
1 : {0,1}; -- new request?
esac;
next(request[2]) :=
case
next(cabin) = 2 : 0; -- satisfied
request[2] : 1; -- continue
1 : {0,1}; -- new request?
esac;
next(request[3]) :=
case
next(cabin) = 3 : 0; -- satisfied
request[3] : 1; -- continue
1 : {0,1}; -- new request?
esac;
init(cabin) := 0;
init(dir) := up;
init(request[0]) := 0;
init(request[1]) := 0;
init(request[2]) := 0;
init(request[3]) := 0;
SPEC
-- spec 1: no deadlock
AG EX 1
-- spec 2: all requests are eventually satisfied?
AG(AF\!request[0] & AF\!request[1] & AF\!request[2] & AF\!request[3])
-- spec 3: requests are eventually all satisfied

b) Results of running with specs

1) Satisfied
2) Satisfied
3) Not satisfied
   The counterexample will show that there are paths where we can always have
   an outstanding request, this is easy to see.