Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

May 26^{th}, 2017

769857918

Name (please print):

Surname (please print):

1 Spin

Model the *Cigarette Smokers* problem **using** the following specification.

Assume that a cigarette requires three ingredients to be made: `TOBACCO`, `PAPER` and `MATCHES`. There are three smokers around a table, each of which has an infinite supply of only **one** ingredient.

Smoker. Each smoker is in a loop waiting for both of his missing ingredients to be put on the *table*. Whenever that happens, he grabs the two ingredients from the table (which becomes empty), rolls a cigarette and smokes it by printing a message. The smoker must also put on the table one unit of his own resource whenever asked to do so through a channel.

Master Agent. Whenever the table is empty, the *master agent* sends a message demanding a unit of resource to be put on the table to two distinct *smokers* using a channel.

Simulate the system and visually verify that it behaves correctly.

2 nuXmv

Model a battery powered *flying drone*, using the skeleton file `/usr/local/docs/drone.smv`, having the following state variables:

- `state`: can be either `OK` or `FAILURE`, the latter meaning that the drone cannot fly any longer
- `power`: ranges from 0 to 100, measures the remaining charge of the drone's battery
- `x`, `y`, `z`: discrete coordinates of the drone; `x` and `y` range in $[-30, 30]$, `z` ranges in $[0, 30]$
- `vx`, `vy`, `vz`: drone's speed vector; `vx` and `vy` range in $[-1, 1]$, `vz` ranges in $[-2, 1]$

Initially, `state` is `OK`, the battery is fully charged and all other variables are equal to 0.

`state`. Assume that the drone is flying in a room surrounded by a concrete wall in all directions, so that if the drone is in the immediate proximity of a wall with a positive speed in the direction of the wall then it will crash against it. The `state` variable of the drone changes to `FAILURE` whenever in the next state of the execution trace the `drone` crashes against a wall. Otherwise, the `state` variable keeps its value.

`power`. The battery is charged back to full state whenever the drone parks at the origin and is in `OK` state. The drone consumes one unit of power when touches the ground but has vertical speed larger than 0. It also consumes one unit of power when it is flying mid-air with a vertical speed larger than -2 (i.e. larger than *free fall* speed). Otherwise, the power level remains unchanged.

`position`. The position of the robot in the next state is obtained by adding its old position vector (x, y, z) with its velocity vector (vx, vy, vz).

`vx`. (resp. `vy`) changes according to this **sorted** set of rules:

- if the drone has crashed over the `x` axis (resp. `y`) or in the next state touches the ground, `vx` (resp. `vy`) is set to 0.
- if the drone is falling, the speed `vx` (resp. `vy`) does not change
- if the drone is powered, it can freely change by one single unit value its current speed
- otherwise, the speed is 0

Express the following properties, and check their expected value with nuXmv:

- **LTL.** if the drone always flies safe then it will remain in good state forever. (true)
- **LTL.** if the drone is flying above ground-level infinitely often, then the drone charges infinitely often too. (true)
- **CTL.** if the drone experiences a collision, then it is necessarily the case that it will eventually hit the ground with negative speed. (true)
- **CTL.** regardless of its position, if the drone is mid-air in `OK` state and has at least 7% of its battery left, then it has at least one possible safe landing strategy. (true)
- **CTL.** write a property s.t. its counter-example is a safe landing strategy for the drone in the state $(0, 0, 30, 0, 0, 0, 5, OK)$ with $(x, y, z, vz, vy, vz, power, state)$.
- **CTL.** write a property s.t. its counter-example is a flight plan that goes through the ordered sequence of states $s_1 = (30, 0, 0, 0, 0, 0, OK)$, $s_2 = (30, 30, 0, 0, 0, 0, OK)$, $s_3 = (30, 30, 30, 0, 0, 0, OK)$ and $s_4 = (20, 30, 30, 1, 0, 0, OK)$ with $(x, y, z, vx, vy, vz, state)$.
- **BONUS.** use `pick_state -s N.NNN` to jump at the last state in the counter-example found for the previous property, and simulate the system with `simulate -iv -k 30`. What happens to the drone? What is the final position of the drone at the end of the simulation?

Skeleton:

```
MODULE main ()
VAR
  ...

DEFINE
  touch_ground := z = 0;
  has_power    := power > 0;
  is_falling   := !touch_ground & (state = FAILURE | !has_power);
  crash_x      := (vx > 0 & x = 30) | (vx < 0 & x = -30);
  crash_y      := (vy > 0 & y = 30) | (vy < 0 & y = -30);
  crash_z      := (vz > 0 & z = 30) | (vz < 0 & z = 0);
  collision    := crash_x | crash_y | crash_z;

ASSIGN

  ...

  next(vz) := case
      -- lift off
      state = OK & touch_ground & has_power      : { 0, 1 };
      -- after crash
      crash_z | (state = FAILURE & touch_ground) : 0;
      -- fall
      is_falling & vz > -2                       : vz - 1;
      is_falling                                 : -2;
      -- powered movement
      has_power & vz >= 1                        : { vz, vz - 1 };
      -- free fall cannot be exploited below height 5 when device is powered on
      has_power & vz <= -2 & z <= 5              : { vz + 1 };
      has_power & vz <= -1 & z <= 5              : { vz, vz + 1 };
      has_power & vz <= -2                       : { vz, vz + 1 };
      has_power                                  : { vz, vz + 1, vz - 1 };
      TRUE                                       : 0;
  esac;

DEFINE
  good_parking  := state = OK & vz >= 0 & touch_ground;
  safe_x_flight := (x > 29 -> vx < 1) & (x < -29 -> vx > -1);
  safe_y_flight := (y > 29 -> vy < 1) & (y < -29 -> vy > -1);
  safe_z_flight := (z > 29 -> vz < 1) & (z < 1   -> vz > -1);
  safe_flight   := safe_x_flight & safe_y_flight & safe_z_flight;
  s1 := (x = 30 & y =  0 & z =  0 & vx = 0 & vy = 0 & vz = 0 & state = OK);
  s2 := (x = 30 & y = 30 & z =  0 & vx = 0 & vy = 0 & vz = 0 & state = OK);
  s3 := (x = 30 & y = 30 & z = 30 & vx = 0 & vy = 0 & vz = 0 & state = OK);
  s4 := (x = 20 & y = 30 & z = 30 & vx = 1 & vy = 0 & vz = 0 & state = OK);

   ...
```