Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

September 20^{th}, 2017

769857918

[COPY WITH SOLUTIONS]

1 Spin

Write a Promela model for an industrial multi-agent system composed by three processes: a "producer", a "labourer" and a "transporter", the latter acting as an intermediary among the two.

- the **producer** sends a "BROKEN" item to the "labourer" through the "transporter", and then waits to receive the item back. The item can be received "BROKEN" if the "labourer" was unable to fix it, or "REPAIRED" otherwise. The "producer" separately counts the number of items that could be repaired and those that could not. After creating and receiving back 100 items, it *checks* that the number of repaired items is larger than the number of items that could not be repaired (through an *assertion*), and *terminates*.
- the **transporter** waits for receiving an item from either the "producer" or the "labourer". Whenever it receives an item from one agent, it forwards it to the other one.
- the **labourer** waits for receiving an item from the "transporter", after which it makes an attempt to repair it before sending it back. The chance that the item is correctly repaired is $\geq 2/3$.

Model the system and use **spin** to check whether the assertion is verified. If not, include the *trail* file with your solution.

Solution:

```
mtype = { BROKEN, REPAIRED };
chan p_t = [0] of { mtype };
chan t_l = [0] of { mtype };

active proctype producer() {
  mtype item;
  byte cc = 100;
  byte repaired, thrown;
  do
    :: cc > 0 ->
       p_t!BROKEN;
       p_t?item;
       if
         :: item == BROKEN -> thrown = thrown + 1;
         :: item == REPAIRED -> repaired = repaired + 1;
       fi;
       cc = cc - 1;
    :: else -> break;
  od
  printf("Repaired: %d -- Thrown: %d\n", repaired, thrown);
  assert(repaired > thrown);
}

active proctype transporter() {
  mtype item;
end:
  do
    :: p_t?item -> t_l!item;
    :: t_l?item -> p_t!item;
  od;
}

active proctype labourer() {
end:
  do
    :: t_l?BROKEN ->
       if
         :: t_l!BROKEN;
         :: t_l!REPAIRED;
         :: t_l!REPAIRED;
       fi
  od;
}
```

2 nuXmv

Model a rechargeable cleaning **robot** which task is to move around a 10×10 room and clean it. Use variables "x" and "y", ranging from 0 to 9, to keep track of the position of the robot and *define* "pos" to be equal $y \cdot 10 + x$. Use a variable "state" with values in { MOVE, CHECK, CHARGE, CLEAN, OFF } to keep track of the next action taken by the robot, and a variable "budget" in { 0..100 } to trace its remaining power. At the beginning, the robot is in state "CHECK" and all other variables are 0.

The robot changes state according to this **ordered** set of rules:

- if the robot is in "pos" 0 and the budget is smaller than 100, then the next state is "CHARGE"
- if the budget is 0, then the next state is "OFF"
- if the robot is in state "CHARGE" or "MOVE", then the next state is "CHECK"
- if the robot is in state "CHECK", then the next state is either "CLEAN" or "MOVE"
- otherwise, the next state is "MOVE".

The budget is decreased by a single unit each time the robot is in state "MOVE" or "CLEAN" (and $budget > 0$), and restored to 100 if the robot is in "CHARGE" state. Otherwise, the budget doesn't change.

Encode, using the constraint-style, the following constraints:

- if the state is different than "MOVE", then the position of the robot never changes.
- if the state is equal to "MOVE", then the robot moves by a single square in one of the cardinal directions: it increases or decreases either "x" or "y", but not both at the same time.

Encode the following properties, and verify with *nuXmv* that the last 4 are `TRUE`:

- in all possible executions, the robot changes position infinitely many times
- it's definitely the case that sooner or later the robot exhausts its budget, turns OFF and stops moving
- it is never the case that the robot's action is either "MOVE" or "CLEAN" and the available budget is zero
- if the robot charges infinitely often, then it changes position infinitely many times
- there exists an execution in which the robot cleans every cell that it visits before leaving it (**note:** you may want to add extra-variables to the ".smv" model in order to encode this property.)
- if the robot is in "pos" 0, then it is necessarily always the case that in the future it will occupy a different position
- the robot does not move along the diagonals

Solution:

```
MODULE main()
VAR
    x       : 0..9;
    y       : 0..9;
    state   : { MOVE, CHECK, CHARGE, CLEAN, OFF };
    budget  : 0..100;
    cleaned : boolean; -- support variable
    prev    : 0..99;   -- support variable

DEFINE
    pos := y * 10 + x;

INIT
    x = 0 & y = 0 & state = CHECK & budget = 0 & cleaned = FALSE & prev = 1;

ASSIGN
    next(state) := case
        pos = 0 & budget < 100          : CHARGE;
        budget = 0                      : OFF;
        state = CHARGE | state = MOVE   : CHECK;
        state = CHECK                   : { CLEAN, MOVE };
        TRUE                            : MOVE;
    esac;

    next(budget) := case
        (state = MOVE | state = CLEAN)
                            & budget > 0 : budget - 1;
        state = CHARGE                   : 100;
        TRUE                             : budget;
    esac;

    next(cleaned) := case
        state = MOVE  : FALSE;
        state = CLEAN : TRUE;
        TRUE          : cleaned;
    esac;

    next(prev) := pos;

TRANS
    state != MOVE -> (x = next(x) & y = next(y));
```

```
TRANS
    state = MOVE -> (next(x) = x + 1 |
                     next(x) = x - 1 |
                     next(y) = y + 1 |
                     next(y) = y - 1);

TRANS
    next(x) != x -> next(y) = y;

TRANS
    next(y) != y -> next(x) = x;

-- in all possible executions, the robot changes position infinitely many times
CTLSPEC AG AF state = MOVE;

-- it's definitively the case that sooner or later
-- the robot exhausts its budget, turns OFF and stops moving
CTLSPEC AF AG budget = 0;
CTLSPEC AF AG state = OFF;
CTLSPEC AF AG state != MOVE;

-- it is never the case that the robot's action is either MOVE or CLEAN
-- and the available budget is zero
CTLSPEC AG ((state = MOVE | state = CLEAN) -> budget > 0);

-- if the robot charges infinitely often, then
-- it changes position infinitely many times
CTLSPEC AG AF (state = CHARGE) -> AG AF (state = MOVE);

-- there exists an execution in which the robot cleans
-- every cell it visits before leaving it
CTLSPEC
    EG ((state=MOVE -> cleaned) & (state=OFF -> cleaned))
    & AG (pos != prev -> AF (state=MOVE | state = OFF));

-- if the robot is in position 0, then it is necessarily always the case that
-- in the future it occupies a different position
CTLSPEC AG (pos = 0 -> AF pos != 0);

-- the robot does not move along the diagonal
INVARSPEC x = next(x) | y = next(y);
```