Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

July 21st, 2016

769857918

[COPY WITH SOLUTIONS]

1 Spin

The 3 Witchers. Write a Promela model for a *Game Theory* scenario in which 200 agents participate to a contest for a prize of a million euro. The contest consists of a single-turn poll during which each agent provides a *secret integer* in the range $[0, 9]$ to a **tracker** process. The *winner(s)* of the contest are those whose value is the closest to the *two thirds* of the average value provided by all the agents.

The voting agents are divided in two groups: 197 **simpleton** processes, which choose a random value in $[0, 9]$ and send it to the tracker, and 3 **witcher** processes which always guess 2. Each agent sends to the tracker both its own *pid* and the integer value. Once collected all the 200 guesses, the **tracker** computes the winning value $(2 * sum/(cc * 3)$, where *sum* is the sum of all guesses and *cc* is the counter of received messages) and prints it to screen. No feedback communication is necessary.

Check with an assertion that 2 is the winning value.

*Is any of these agents **rational**? why? (No evaluation)*

Solution:

```
chan pool = [200] of { int, int };

active [3] proctype witcher () {
  pool ! _pid(2);
}

active [197] proctype simpleton() {
  int r;
  if
    :: r = 0;
    :: r = 1;
    :: r = 2;
    :: r = 3;
    :: r = 4;
    :: r = 5;
    :: r = 6;
    :: r = 7;
    :: r = 8;
    :: r = 9;
  fi;
  pool ! _pid(r);
}

active proctype tracker() {
  int v, sum, id, cc;
  do
    :: pool? id(v) ->
       sum = sum + v;
       cc = cc + 1;
    :: cc == 200 ->
       break;
  od;
  sum = 2 * sum / (cc * 3);
  printf("Best value is: %d\n", sum);
  assert(sum == 2);
}
```

2 nuXmv

Blind Hunt. Model a blind *mouse* which lives in a 7×7 room. The *mouse* takes as input four arguments: "room", "cheese", "cat" and "tile". Use variables "x" and "y", both ranging from 0 to 6, to keep track of the position of the *mouse* and **define** "pos" to be equal $y \cdot 7 + x$. Use a variable "state" with values in { IDLE, HUNGRY, SLEEPY, DEAD } to keep track of the mouse state. Define "safe_spot" to be equal to "tile", and its corresponding coordinates "ss_x" and "ss_y" to be equal to "tile - ss_y * 7" and "tile / 7" respectively. Initially, the mouse is in "IDLE" state and located in his `safe_spot`. Then, model the evolution of the *mouse* state with the following ordered list of rules:

- if the *mouse* is "DEAD" or its next location coincides with that of the *cat*, then in the next state the *mouse* is "DEAD"
- if the *mouse* is in "IDLE" state, then it can remain "IDLE" or become "HUNGRY" at random
- if in the next state the location of the *mouse* and *cheese* coincide, then the *mouse* becomes "SLEEPY"
- if the next location of the *mouse* is its "safe_spot" and the *mouse* is not "HUNGRY", then the *mouse* becomes "IDLE"
- otherwise, the *mouse* keeps its current state

Encode, using the constraint style, the following requirements (you are allowed to split each constraint in several sub-formulas):

- if the *mouse* is either "HUNGRY" or "SLEEPY", then it can either remain on the same location or move by a single square in one of the cardinal directions: it increases or decreases either "x" or "y", but not both
- if the *mouse* is either "IDLE" or "DEAD", then it remains in the same location
- if the *mouse* is "HUNGRY", then in the next state its distance to the *cheese* will be *strictly smaller* than it is now[1].
- if the *mouse* is "SLEEPY", then either in the next state its distance to its *safe_spot* is *strictly smaller* than it is now[2] or the *mouse* is already located in the *safe_spot* and thus its position will not change

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

- if the *mouse* does never trespass the safe zone boundary[3] the *mouse* will never die
- if the *cheese* is not always placed within the safe zone boundary, then there exists a future in which the *mouse* is "DEAD"
- if –at any given time– the *mouse* is hungry, then it will eventually eat the cheese (reach its location), unless it dies in the meanwhile
- if the *mouse* is hungry infinitely often, then the *mouse* is sleepy infinitely often
- there exists an execution in which the *cheese* location never changes
- the *mouse* does not move along the diagonal
- the *cat* moves only along the diagonal

Remark: use the file *the_blind_hunt.smv* located in `/usr/local/docs` as starting point.

[1] Given X, Y, X', Y', which are the current and the next coordinates of the *mouse* respectively, and X_c, Y_c, the current coordinates of the *cheese*, then the following inequality must hold: $(X' - X_c)^2 + (Y' - Y_c)^2 < (X - X_c)^2 + (Y - Y_c)^2$

[2] Same formula as above, with ss_x, ss_y instead of X_c, Y_c

[3] the safe zone corresponds to any tile s.t. $2 \leq x \leq 4 \wedge 2 \leq y \leq 4$

Solution:

```
MODULE room(in_width, in_height)
DEFINE
  width  := in_width;
  height := in_height;


MODULE cheese(room, cat, mouse, tile)
VAR
  x : 0..(room.width - 1);
  y : 0..(room.height - 1);
DEFINE
  pos := y * room.width + x;
INIT
  pos = tile;
INVAR
  pos != mouse.safe_spot & pos != cat.pos;
TRANS
  (mouse.pos = pos -> next(pos) != pos) &
  (mouse.pos != pos -> next(pos) = pos);


MODULE cat(room, cheese, mouse, tile)
VAR
  x : 0..(room.width - 1);
  y : 0..(room.height - 1);
INIT
  pos = tile;
DEFINE
  pos := y * room.width + x;
INVAR
  pos != mouse.safe_spot & pos != cheese.pos;
INVAR
  (x > 4) | (x < 2) | (y > 4) | (y < 2);
TRANS
  (next(x) = x + 1 | next(x) = x - 1) & (next(y) = y + 1 | next(y) = y - 1);


MODULE mouse(room, cheese, cat, tile)
VAR
  x     : 0..(room.width - 1);
  y     : 0..(room.height - 1);
  state : { IDLE, HUNGRY, SLEEPY, DEAD };
DEFINE
  pos := y * room.width + x;
```

```
  safe_spot := tile;
  ss_x := (tile - ss_y * room.width);
  ss_y := (tile / room.width);
INIT
  pos = safe_spot & state = IDLE;
ASSIGN
  next(state) := case
    next(pos) = next(cat.pos) | state = DEAD : DEAD;
    state = IDLE                             : { IDLE, HUNGRY };
    next(pos) = cheese.pos                   : SLEEPY;
    next(pos) = safe_spot & state != HUNGRY  : IDLE;
    TRUE                                     : state;
  esac;
TRANS
  (state = HUNGRY | state = SLEEPY)
    -> ((next(x) = x + 1 | next(x) = x - 1 | next(x) = x) &
        (next(y) = y + 1 | next(y) = y - 1 | next(y) = y));
TRANS
  (next(x) != x -> next(y) = y) & (next(y) != y -> next(x) = x);
TRANS
  (state = IDLE | state = DEAD) -> (next(x) = x & next(y) = y);
TRANS
  state = HUNGRY -> (
  (next(x) - cheese.x) * (next(x) - cheese.x) + (next(y) - cheese.y) * (next(y) - cheese.y
  < (x - cheese.x) * (x - cheese.x) + (y - cheese.y) * (y - cheese.y));
TRANS
  state = SLEEPY -> (
  (next(x) - ss_x) * (next(x) - ss_x) + (next(y) - ss_y) * (next(y) - ss_y)
  < (x - ss_x) * (x - ss_x) + (y - ss_y) * (y - ss_y)
  | (pos = safe_spot & next(pos) = pos));


MODULE main()
VAR
  room   : room(7, 7);
  cheese : cheese(room, cat, mouse, 0);
  cat    : cat(room, cheese, mouse, 48);
  mouse  : mouse(room, cheese, cat, 24);


-- if the mouse does never trespass the safe zone boundary,
-- the mouse will never die
LTLSPEC (G (mouse.x >= 2 & mouse.x <= 4 & mouse.y >= 2 & mouse.y <=4))
        -> G mouse.state != DEAD;
```

```
-- if the cheese is not always placed within the safe zone boundary,
-- then there exists a future in which the mouse is ‘‘DEAD’’
CTLSPEC EF (cheese.x > 4 | cheese.x < 2 | cheese.y > 4 | cheese.y < 2)
        -> EF mouse.state = DEAD;

-- if --at any given time-- the mouse is hungry,
-- then it will eventually eat the cheese (reach its location),
-- unless it dies in the meanwhile
LTLSPEC G (mouse.state = HUNGRY -> F (mouse.pos = cheese.pos | mouse.state = DEAD));

-- if the mouse is hungry infinitely often, then
-- the mouse is sleepy infinitely often
LTLSPEC G F (mouse.state = HUNGRY) -> G F (mouse.state = SLEEPY);

-- there exists an execution in which the cheese location never changes
CTLSPEC EG cheese.pos = 0;

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

-- the cat only moves along the diagonal
INVARSPEC cat.x != next(cat.x) & cat.y != next(cat.y);
```