Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

January 11^{th}, 2018

769857918

[COPY WITH SOLUTIONS]

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. The *master agent* chooses the *smokers* that have to put their own resource on the table using a *uniform random distribution*.

Simulate the system and visually verify that it behaves correctly: the simulation output consists of an infinite execution trace in which each **smoker** smokes infinitely often.

Solution:

```
#define TOBACCO 1
#define PAPER   2
#define MATCHES 4

unsigned table : 3 = 0;

inline smoke()
{
    if
      :: id & TOBACCO -> printf("Tobacco Smokes\n");
      :: id & PAPER   -> printf("Paper Smokes\n");
      :: id & MATCHES -> printf("Matches Smokes\n");
    fi
    table = 0;
};

proctype smoker(mtype id; chan master)
{
  do
    :: atomic {
        master?eval(id) ->                          assert(~(id & table));
        table = (table|id);                         assert(id&table);
       }
    :: table == ((TOBACCO|PAPER|MATCHES) & ~id) ->
        smoke();                                    assert(~table);
  od
}

init
{
  unsigned i : 2 = 0; unsigned j : 2 = 0;
  chan master = [0] of { mtype };

  run smoker(TOBACCO, master);
  run smoker(PAPER, master);
  run smoker(MATCHES, master);

  do
    :: table == 0 ->
        select(i: 0..2); select(j: 0..2);

        if
          :: i == j -> j = (j + 1) % 3;
          :: i == j -> j = (j + 2) % 3;
          :: else -> skip;
        fi;

        master!(1<<i); master!(1<<j);
  od;
}
```

2 nuXmv

Model a simple **alarm** system positioned inside the **safe** of a bank. The **alarm** system can be activated and de-activated using a `pin`. After being activated, the **alarm** system enters a `waiting` period of 10 seconds, time that allows users to evacuate the **safe**, after which the **alarm** is `armed`. The **alarm** detects and `intrusion` when someone is inside the **safe** and the alarm is `armed`, after which it enters a `waiting` period of 5 seconds to allow the intruder to de-activate the alarm using the `pin`. If the alarm is not de-activated after an intrusion is detected, it will `fire` and remain `fired` until de-activation.

The alarm system is comprised by a `state` variable, with domain { `OFF`, `EVACUATE`, `ARMED`, `INTRUSION`, `FIRED` }, and a `s_clock` variable, with domain equal to 0..59. Initially, `state` is `OFF` and `s_clock` is 0.

The alarm system has two `boolean` inputs: `sensor` –`true` iff a person is detected inside the safe– and `use_pin` –`true` iff the pin is being used–. Express the fact that a person must be inside the safe to use the pin as an *invariant* of the inputs.

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

- if the state is `OFF` and the pin is used, then the next state is `EVACUATE`
- if the pin is used, then the next state is `OFF`
- if the state is `EVACUATE` and the internal clock is 0, then the next state is `ARMED`
- if the state is `ARMED` and a person is detected in the safe, then the next state is `INTRUSION`
- if the state is `INTRUSION` and the internal clock is 0, then the next state is `FIRED`
- otherwise, the state does not change

The value of `s_clock` is set to 10 when the `state` value changes from `OFF` to `EVACUATE`, and it is set to 5 when the `state` value changes from `ARMED` to `INTRUSION`. Otherwise, its value is decreased by one unit at each transition until it reaches 0.

Encode the following *LTL* properties, and verify with *NuSMV* that they are `true`:

- if the input pin is never used, then the alarm state is always `OFF`
- it is always true that, whenever an intrusion is detected then sooner or later the alarm state will be either `OFF` or `FIRED`
- it is always true that "if the alarm is armed in a certain state s_k, but the pin is never used starting from s_k onward, then it is necessarily the case that either the sensor won't detect any intruder (starting from s_k onward) or the alarm will eventually fire"
- if the state of the alarm is infinitely often equal to `EVACUATE`, then someone must enter the safe infinitely often

Solution:

```
MODULE main()
VAR
    sensor : boolean;
    pin    : boolean;
    a      : alarm(sensor, pin);
INVAR
    pin -> sensor;

LTLSPEC ( G !pin) -> ( G a.state = OFF);
LTLSPEC G ((a.state = INTRUSION) -> F (a.state in { FIRED, OFF }));
LTLSPEC G ((a.state = ARMED & G !pin) ->
           ( G !sensor | F a.state = FIRED));
LTLSPEC ( G F a.state = EVACUATE) -> ( G F sensor );

MODULE alarm(sensor, use_pin)
VAR
    state   : { OFF, EVACUATE, ARMED, INTRUSION, FIRED };
    s_clock : 0..59;

INIT
    state = OFF & s_clock = 0;

ASSIGN
    next(state) := case
        OFF = state & use_pin           : EVACUATE;
        use_pin                         : OFF;
        EVACUATE = state & s_clock = 0  : ARMED;
        ARMED = state & sensor          : INTRUSION;
        INTRUSION = state & s_clock = 0 : FIRED;
        TRUE                            : state;
    esac;

TRANS
case
    (state = OFF & next(state) = EVACUATE)    : next(s_clock) = 10;
    (state = ARMED & next(state) = INTRUSION) : next(s_clock) = 5;
    (s_clock > 0) : next(s_clock) = s_clock - 1;
    TRUE : next(s_clock) = 0;
esac;
```