Formal Methods in Computer Science

Thanh-Hai Tran and Florian Zuleger

Exercise Sheet – Block 4
Due January 13, 2020

1 Overview

This exercise sheet is connected to Block 4 (Satsifiability Model Checking). It focuses on practical aspects of model checking and requires using the **NuSMV symbolic model checker** developed at FBK-IRST, Trento.

As a prerequisite, please get your own copy of NuSMV from:

`http://nusmv.fbk.eu/`

An introduction to NuSMV was given in the exercise session on December 17, 2019. The slides used in that exercise session are available on TUWEL.

It is also recommended, but not required, to read a tutorial on NuSMV available at:

`http://nusmv.fbk.eu/NuSMV/tutorial/`

2 Exercise Sheet - Problems

There are three exercises, yielding a total of 15 points.

Exercise 1. (2+2 points) Consider the following NuSMV program implementing mutual exclusion between two processes.

```
MODULE main
VAR
   semaphore : boolean;
   proc1 : process user(semaphore);
   proc2 : process user(semaphore);
ASSIGN
   init(semaphore) := FALSE;

MODULE user(semaphore)
VAR
   state : {idle, entering, critical, exiting};
ASSIGN
   init(state) := idle;
   next(state) :=
               case
                   state = idle                  : {idle, entering};
                   state = entering & !semaphore : critical;
                   state = critical              : {critical, exiting};
                   state = exiting               : idle;
                   TRUE                          : state;
               esac;
```

```
   next(semaphore) :
                case
                    state = entering : TRUE;
                    state = exiting  : FALSE;
                    TRUE             : semaphore;
                esac;
FAIRNESS
running
```

This NuSMV program uses the variable `semaphore` to implement mutual exclusion between the two processes `proc1` and `proc2`. Each process has four states: `idle`, `entering`, `critical` and `exiting`. The `entering` state indicates that the process wants to enter its critical region. If the variable `semaphore` is `FALSE`, it goes to the `critical` state, and sets `semaphore` to `TRUE`. On exiting its critical region, the process sets `semaphore` to `FALSE` again.

(a) A safety property P of this program is that "it should never be the case that the two processes `proc1` and `proc2` are at the same time in the `critical` state". Express P as a *CTL formula* and add it as a CTL specification to the above NuSMV program. Verify the CTL specification by running NuSMV on the annotated program. If NuSMV produces a counterexample, explain the counterexample!

(b) A liveness property Q of this program is that "whenever process `proc2` wants to enter its `critical` state, it eventually does". Express Q as an *LTL formula* and add it as a LTL specification to the above NuSMV program. Verify the LTL specification by running NuSMV on the annotated program. If NuSMV produces a counterexample, explain the counterexample!

Submission guidelines: For each taks, submit the annotated NuSMV program together with the output of running NuSMV on it. In case a counterexample was generated, submit your narrative explanation of the counterexample.

Exercise 1.2. (2+2 points) Consider the following NuSMV program implementing a simple, deterministic counter modulo 8.

```
MODULE main
VAR
  y : 0..15;

ASSIGN
  init(y) := 0;

TRANS
  case
    y = 7    :  next(y) = 0;
    TRUE     :  next(y) = (y + 1) mod 16;
  esac
```

(a) Consider the property expressing that "there is a value of y whose next value is 8". Express this property as an LTL formula and add it as a LTL specification to the above NuSMV program. Verify the LTL specification by running NuSMV on the annotated program. If NuSMV produces a counterexample, explain the counterexample!

(b) Consider the property expressing that "there is a value of y whose next value is 7". Express this property as an LTL formula and add it as a LTL specification to the above NuSMV program. Verify the LTL specification by running NuSMV on the annotated program. If NuSMV produces a counterexample, explain the counterexample!

Submission guidelines: For each taks, submit the annotated NuSMV program together with the output of running NuSMV on it. In case a counterexample was generated, submit your narrative explanation of the counterexample.

Exercise 1.3. (1+4+2 points) Consider the following puzzle that is an instance of the puzzle known as the "Tower of Hanoi".

There are three poles (`left`, `middle`, `right`) and four ordered disks `d1`, `d2`, `d3`, `d4` of different sizes, with disk `d1` being the biggest one. Initially, all four disks are on the `left` pole in ascending order, the smallest at the top. The goal of the puzzle is to move all four disks to the `right` pole, using the following simple rules:

- Only one disk can be moved at a time;
- Each move consists of taking the upper disk from one of the poles and placing it on top of another pole;
- No disk may be placed on top of a smaller disk.

The NuSMV program below describes the skeleton of a Hanoi tower puzzle with four disks. The skeleton declares the state variables of the puzzle and defines macros for moving a disk.

```
MODULE main
-- Hanoi tower with three poles (left, middle, right)
-- and four ordered disks d1, d2, d3, d4,
-- disk d1 is the biggest one
VAR
  d1 : {left,middle,right};
  d2 : {left,middle,right};
  d3 : {left,middle,right};
  d4 : {left,middle,right};
  move : 1..4; -- possible moves
DEFINE
  move_d1 := move=1;
  move_d2 := move=2;
  move_d3 := move=3;
  move_d4 := move=4;

-- di is on top of a pole iff di!=dj for every j>i

  top_d1 :=
           d1!=d2 &
           d1!=d3 &
           d1!=d4;
  top_d2 :=
           d2!=d3 &
```

```
          d2!=d4;
top_d3 :=
          d3!=d4;
top_d4 := TRUE;
```

Complete the program skeleton above to model the puzzle, ensuring that the puzzle yields a solution (that is, all four disks are on the `right` pole). Your tasks are as follows:

(a) Declare the set of initial states;

(b) Formalize the transition relation for the existing variables;

(Hint: declare the transition relation by completing and continuing the following skeleton `TRANS move_d1 -> ...`)

(c) Formalize in CTL the requirement that the puzzle has a solution and make sure that your design satisfies it.

Submission guidelines: Submit the NuSMV program completed with initial states and transition relation and annotated with the proper CTL specification. Submit and explain the output of running NuSMV on your solution.