UPPSALA UNIVERSITET
Matematiska Institutionen
Anton Hedin

LABORATORY EXERCISE 2
APPLIED LOGIC
2009-12-08

CTL Model Checking with NuSMV

The first part of the laboratory exercises is a brief introduction to the software NuSMV. The second part consists of a couple of more involved problems. You may work in pairs. Hand in finished and annotated files at the latest January 20th 2010.

NuSMV is a BDD-based (Binary Decision Diagram) model checker that allows to check finite state systems against specifications in the temporal logic CTL. The software is freely available at `http://nusmv.irst.itc.it/` where you will also be able to find a tutorial and manual.

Part 1

We start with some simple examples: Run NuSMV by typing `NuSMV -int` in a terminal (the argument `-int` opens the program in interactive mode) and check some of the commands by typing `help [command]` (simply typing `help` brings up a list of commands)

```
$ NuSMV -int
*** This is NuSMV 2.4.0 (compiled on Tue Aug  1 07:44:56 GMT 2006)
*** For more information on NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.
[...]
NuSMV > help
[...]
which                   write_boolean_model       write_flat_model
write_order
NuSMV > help read_model

  read_model - Reads a NuSMV file into NuSMV.
[...]
NuSMV > help go

  go - Initializes the system for the verification.
[...]
NuSMV > quit
```

Exercise 1 (warm-up) In the examples folder (`NuSMV/share/nusmv/examples/smv-dist`) there is a file called `short.smv` (you can also find it on the webpage under "NuSMV Examples"). See also p. 192-193 in the course book LCS for a description of the code.

```
MODULE main
VAR
  request : {Tr, Fa};
  state : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
                   state = ready & (request = Tr): busy;
                   1 : {ready,busy};
                 esac;
SPEC
  AG((request = Tr) -> AF state = busy)
```

Open the file in interactive mode by typing `NuSMV -int short.smv` in the folder containing the file. Type the command `go` to initialize the system for verification. Verify the specification given in the file by using the command `check_ctlspec`. You can also check specifications directly in the terminal: type `help check_ctlspec` and figure out how to check the formula $\mathsf{AG}(\texttt{request} = \texttt{Tr} \rightarrow \mathsf{AX}(\texttt{state} = \texttt{busy}))$ in this way. Test some other CTL formulas.

Exercise 2 Open the file `counter.smv` in interactive mode (located in the same folder as `short.smv`). This is a system description given by several modules. The module `counter_cell` has one parameter and is called by the main module (a NuSMV program must always contain a module named main, without any parameters).

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
  AG AF bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := value + carry_in mod 2;
```

```
DEFINE
  carry_out := value & carry_in;
```

What does the code do? To verify that you understand the code you can perform a simulation of the system. This is done by first selecting a state among the initial states

```
NuSMV > go
NuSMV > pick_state -i
```

Then a simulation, of k steps, starting from the chosen state is performed by

```
NuSMV > simulate -i k
```

The system is deterministic so it should be easy to verify your initial guess. As before, you can verify CTL formulas with the command `check_ctlspec -p "CTL-formula"`. Try for example AX(`bit0.value=0`), ¬EF(`bit0.value=1`∧ `bit1.value=0` ∧ `bit2.value=1`) and A[`bit2.value=0` U `bit2.value=1`].

Exercise 3 An asynchronous system. Consider the following situation: Adam and Eve are sitting at a table to enjoy a nice meal. They each have a plate of food, but there is only one fork at the table. Thus only one of them can be eating at any given time. To make sure that both will be able to finnish their meal there is a waiter that controls acces to the fork. Adam and Eve may at any time request to use the fork, and if it is available the person requesting it should have it. This situation can be modelled by the following code. The main module represents the waiter and determines whose turn it is, based on received requests. Your first task is to fill in the gaps in the code.

```
MODULE proc(other-person,turn,myname)
VAR
    status: {no-fork,request-fork,eating};
ASSIGN
    init(status) := [...]; -- complete here
    next(status) :=
          case
              (status = request-fork) &
              (other-person=request-fork) &
              (turn=myname) : eating;
              (status = request-fork) & -- Complete here
              (status = no-fork) -- Complete here
              (status = eating) : {eating,no-fork};
              1 : status;
          esac;
```

```
FAIRNESS running

MODULE main
VAR
    Adam : process proc(Eve.status,turn,0);
    Eve : process proc(Adam.status,turn,1);
    turn : boolean;
ASSIGN
    init(turn) := 0;
-- You can leave the evolution empty:
-- in this way turn changes randomly.
```

Now load the model and formalize and check the following properties

- Adam and Eve are never able to eat at the same time (safety),
- If Adam (Eve) requests the fork he (she) should eventually be allowed to eat (liveness).

A *fairness* constraint restricts the attention only to fair execution paths. When evaluating specifications, the model checker considers path quantiers to apply only to fair paths. The fairness condition `running` restricts attention to paths along which the module in which it appears is selected for execution infinitely often. Try to find a fairness condition making the second formula above true (see the example on mutual exclusion in the course book).

Now check that the property of *No Strict Sequencing* holds: Adam and Eve do not have to take turn eating. E.g. Eve can request the fork, eat, lay down the fork, request it and start eating again without Adam eating in between. Hint: see p. 215 in the course book.

Part 2

Now that you are acquainted with NuSMV it is time for something a bit more involved.

Exercise 4 Write a code modelling a toy elevator system. The elevator is in a building with 3 floors, each floor has one button for calling the elevator. When the button on one floor is pressed, and no other button is already pressed, the elevator should start travelling to the floor with the button pressed. If other buttons are pressed during this time you may assume this has no effect, i.e. you do not have to set up a queue (this is for example the way elevator 6 in the Ångström building works :-)). Use the simulation mode to convince yourself that your model is correct.

Formalize and check that your model satisfies the following properties.

- If the elevator is on floor i and no button is pressed then there is the possibility that it stays on floor i with no button being pressed forever.
- If the button on floor i is pressed (when it gives effect) the elevator should eventually reach floor i.
- The elevator can never travel two floors in one time step.

Make your model satisfy the following property.

- The elevator visits every floor infinitely often.

Does the first property still hold?

Exercise 5 Use NuSMV to find a solution to the classical puzzle "The towers of Hanoi". It consists of three poles (left, right and middle) and 4 disks (1, 2, 3, 4) of different size. Initially all discs are placed on the left pole. The goal is to move all disks from the left pole to the right pole (using the middle pole), with the following constraints: only one disc can be moved at a time; a disc must never be placed on top of a smaller one and a disc can only be moved if there are no other disks on top of it.

Hint: the disks can be described as variables ranging over the set {left, right, middle}, use this to code a module describing possible moves in the puzzle. The solution can then be given as a counter example to the truth of a certain CTL formula (compare with the ferryman problem on p. 199 in the course book).