Automated Reasoning 2015-16: Coursework 2

Model Checking with NuSMV

Deadline: Friday 18 March 2016, 4pm

1 Getting Started

Create a new directory for your work on a DICE machine and change to that directory. Download the template files from the coursework section of the course home page

`http://www.inf.ed.ac.uk/teaching/courses/ar`

For instructions on using NuSMV, see the *NuSMV Startup Guide* linked to from this section.

2 Preliminary Exercises

2.1 LTL (15%)

Create a NuSMV model for the transition system shown in Figure 1.

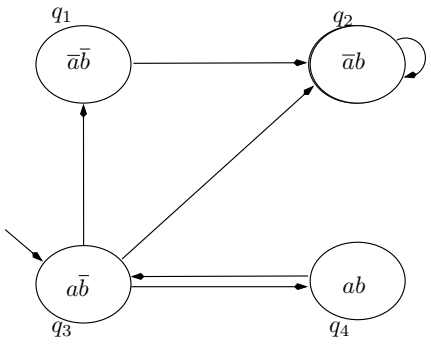

Figure 1: Transition system for LTL Exercise

For each of the LTL formulas ϕ below,

1. $\mathbf{G}\, a$
2. $a\, \mathbf{U}\, b$
3. $a\, \mathbf{U}\, \mathbf{X}\, (a \wedge \neg b)$
4. $\mathbf{X}\, \neg b \wedge \mathbf{G}\, (\neg a \vee \neg b)$
5. $\mathbf{X}\, (a \wedge b) \wedge \mathbf{F}\, (\neg a \wedge \neg b)$

use NuSMV to (i) determine whether this model satisfies the formula ϕ, and (ii) persuade NuSMV to exhibit some path in the model that satisfies ϕ.

Hints:

- It's simplest to create a NuSMV model of the state machine that uses 1 state variable with 4 values, one for each of the states of the state machine. Then use DEFINE assignments to specify in which states the atomic propositions 'a' and 'b' are true. An alternative approach that can yield a more compact model, but that can be slightly less straightforward, is to introduce 2 state variables, one for 'a', one for 'b'.
- For (ii), consider what NuSMV does if you direct it to try proving $\neg\phi$.

Check that the answers you get with NuSMV correspond to your own understanding of the model and the formulas.

Insert your answers into template file `ltl-exercise.smv`. At the top of this file you insert your model and a brief explanation of the approach you use for finding satisfying paths. Then, for each part of the question, you give the NuSMV code for the LTL formula, state whether the formula is valid, and give an example satisfying path.

2.2 CTL (20%)

Which of the following pairs of CTL formulas are equivalent? For those which are, argue briefly why they are equivalent. For those which are not, create a NuSMV file with a model and the two formulas, each as a property to check, such that one property is true of the model and the other false. Use the `CTLSPEC` keyword in NuSMV to introduce CTL properties, just as the `LTLSPEC` keyword introduces LTL properties.

1. $\mathbf{EF}\,\phi$ and $\mathbf{EG}\,\phi$
2. $\mathbf{EF}\,\phi \vee \mathbf{EF}\,\psi$ and $\mathbf{EF}\,(\phi \vee \psi)$
3. $\mathbf{AF}\,\phi \vee \mathbf{AF}\,\psi$ and $\mathbf{AF}\,(\phi \vee \psi)$
4. $\mathbf{AF}\,\neg\phi$ and $\neg\mathbf{EG}\,\phi$
5. $\mathbf{EF}\,\neg\phi$ and $\neg\mathbf{AF}\,\phi$
6. $\mathbf{A}[\phi_1\ \mathbf{U}\ \mathbf{A}[\phi_2\ \mathbf{U}\ \phi_3]]$ and $\mathbf{A}[\mathbf{A}[\phi_1\ \mathbf{U}\ \phi_2]\ \mathbf{U}\ \phi_3]$.
7. $\top$ and $\mathbf{AG}\,\phi \Rightarrow \mathbf{EG}\,\phi$
8. $\top$ and $\mathbf{EG}\,\phi \Rightarrow \mathbf{AG}\,\phi$

Collect all your answers together in supplied template file `ctl-exercise.smv`.

3 Verifying a Lift Controller

3.1 Description of provided lift system model

The provided file `lift.smv` has a NuSMV model of a lift system.

The module `lift` models the state of the lift itself: it tracks which of 4 floors the lift is positioned at, and whether the lift is moving or not. This module has two inputs, `go_up` and `go_down` for commanding the lift to move up and down respectively. The state machine for

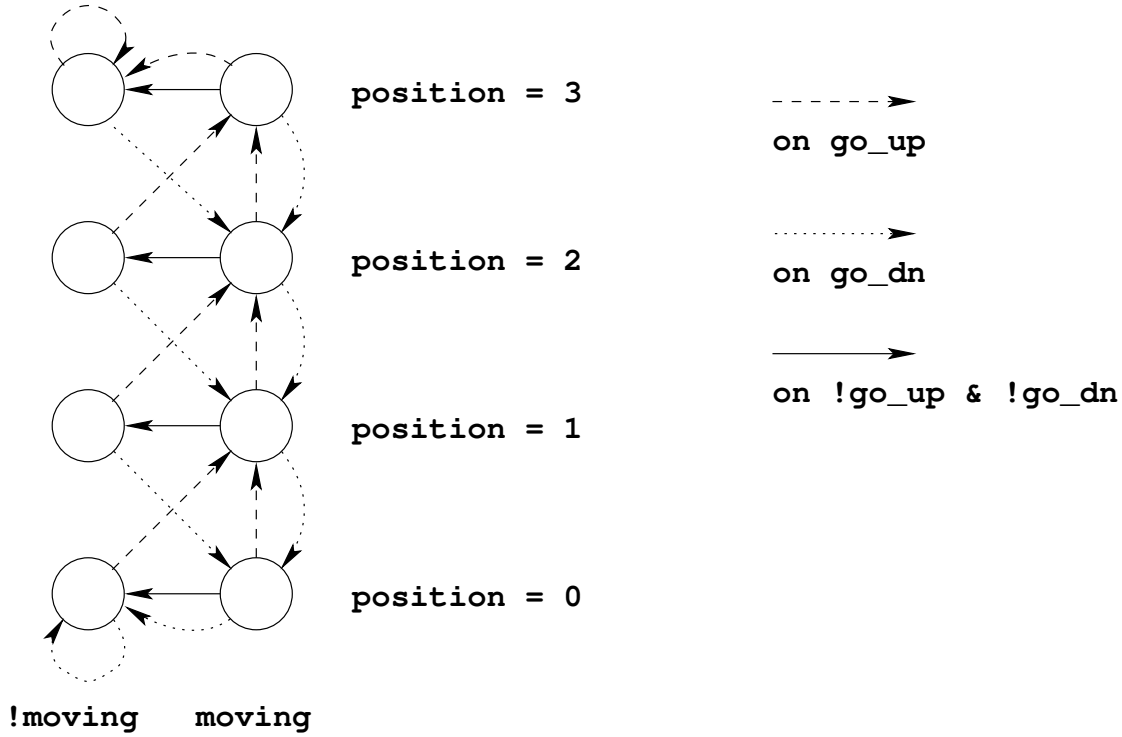

Figure 2: Lift state machine

the lift is shown in Figure 2.

There are various buttons controlling the operation of the lift. Inside the lift are *floor* buttons for requesting the lift to go to each floor, and outside the lift on each floor are *up* and *down* buttons for requesting the lift for going up or down. However, as expected, there is no *up* button on the top floor, or *down* button on the bottom floor. For simplicity, we don't have explicit state variables tracking the pressing of each button. Rather, we have introduced for each button an instance of a module `button`, which has a single boolean state variable `active`. A button starts as inactive and can non-deterministically either choose to stay inactive or become active. This models someone pressing the button. Once active, a button can only become inactive if it receives a `reset` input from the lift controller.

The `main` module models the lift controller and also contains an instance of the `lift` module and appropriate instances of the `button` module. The lift controller takes as input the state of the lift and buttons and generates outputs to control the lift and reset the buttons. The implemented control strategy is a fairly straightforward one: the controller alternates between *going-up* and *going-down* modes. When it is in the *going-up* mode, it

repeatedly services requests from *up*, *down* and *floor* buttons above its current position. When it is in the *going-down* mode, it repeatedly services requests from *up*, *down* and *floor* buttons below its current position. In addition, the lift can be an *idle* mode, for when there are no requests for it to move up or down. The mode of the controller is tracked by a state variable `mode`. The state machine for the controller is shown in Figure 3.

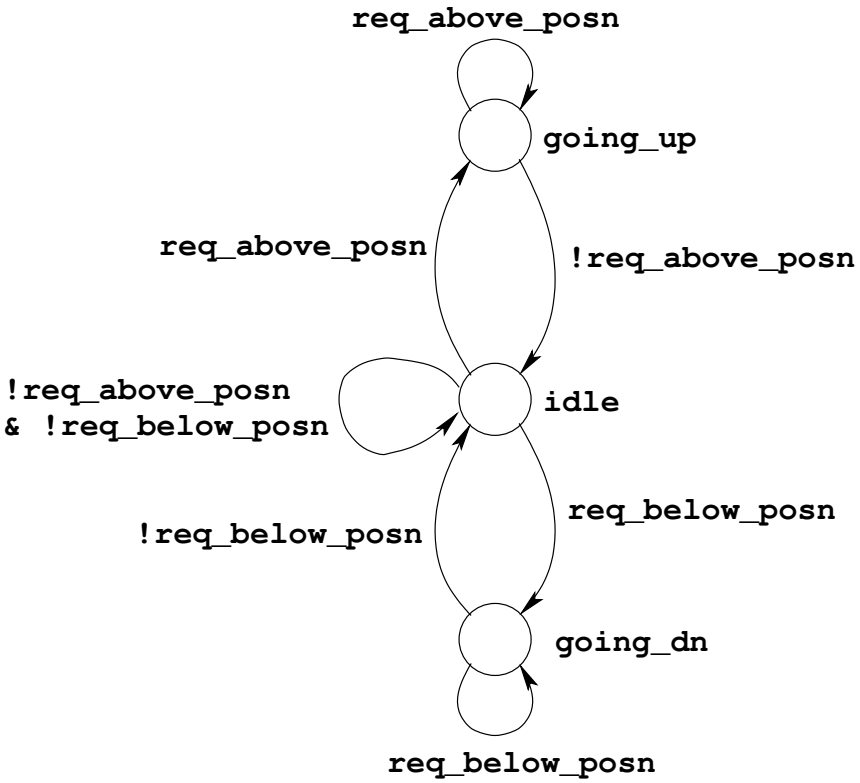

Figure 3: Lift Controller FSM

3.2 Properties to verify (25%)

In the provided template file `lift-properties.smv`, write formulas for the following properties. Add `DEFINE` assignments at the start of this file to introduce new variables to help keep your formulas concise and improve their readability. Verify your properties with NuSMV by running the command

```
NuSMV -pre cpp lift.smv
```

The `lift.smv` file brings in the `lift-properties.smv` file using a preprocessor `#include` directive at its end. The `-pre cpp` option to NuSMV here is necessary to ensure it runs the C preprocessor on `lift.smv` in order to interpret this directive.

1. Write LTL formulas for:

   (a) The controller never commands the lift to go up and go down simultaneously. This is an example of a *safety* property.

(b) Whenever there is a request from a button on a different floor from the current position, eventually the lift is stopped (not moving) at that floor.

This is an example of a *liveness* property.

(c) Whenever the lift is stopped at floor 0 and the floor buttons for floors 1 and 2 are active, the lift will stop first at floor 1 and second at floor 2.

(d) Whenever the lift is stopped at floor 0 and the down buttons for floors 1 and 2 are active, the lift will stop first at floor 2 and second at floor 1.

We don't expect this property to be true; for example, if the floor button for floor 3 is active, we expect a stop at floor 3 first. Add to your LTL formula a minimal set of extra conditions about other buttons not being active under which you would expect this property to be true. Whereas you will find that the previous LTL properties are true for the provided lift system, you will find this property is false because of a bug in the controller. In Section 3.3 you explore and fix this bug.

2. Write CTL formulas for:

(a) It is possible for the lift to go to floor 3 and stay stopped on floor 3 for ever.

(b) At all reachable states, it is possible for the lift to return to being stopped at floor 0.

3.3 Controller bug to fix (20%)

The controller has a bug. In this part you discover and fix it.

1. In the indicated place in `lift-properties.smv`, write an LTL property that checks that, if none of the *up*, *down* or *floor* buttons for floor 2 are ever active, the lift never stops at floor 2.

   NuSMV should find it false and show a counter-example.

2. The counter-examples returned by NuSMV are not always the shortest. To find the shortest, use the *bounded-model-checking* (BMC) capabilities of NuSMV.

   By default NuSMV uses a sound and complete algorithm based on BDD-based techniques to check temporal logic formulas. However it also implements an alternate BMC algorithm which makes use of boolean satisfiability checkers (SAT solvers) such as `MiniSat` and `zchaff`. BMC involves searching for counter-examples to an LTL formula up to a given size (bound). BMC is an unsound, but complete technique. If it finds a counter-example the counter-example is real, but it may fail to find a counter-example just because there is none shorter than the given bounds. BMC is very useful, as it can often handle much larger models than BDD-based model checking.

   To use BMC, enter for example

   ```
   NuSMV -pre cpp -bmc -bmc_length 10 -n 6 lift.smv
   ```

Here, the option `-bmc_length 10` tells NuSMV to search for counter-examples of up to size 10 and the option `-n 6` tells NuSMV to check just the 6th property (counting from 0) in the `lift-properties.smv` file. If you haven't changed the ordering, this should be the number of the property you write for this question.

Give a summary of the behaviour found in the shortest counter-example in the indicated place in the `lift-properties.smv` file.

3. Make a copy of `lift.smv` called `lift-fixed.smv`. Make changes to the lift controller code in the `main` module in `lift-fixed.smv` file to fix this bug. Your changes should address the general problem identified by this bug. Full marks will not be given if you just make some minimal change such that the particular property you wrote to identify the problem now checks true.

   Do *not* alter `lift.smv`.

   At the top of `lift-fixed.smv`, add comments briefly describing your diagnosis of the problem and why your changes fix it.

3.4 Principles of LTL model checking (20%)

As remarked in lecture, in LTL model checking of a formula ϕ, one constructs a Büchi automaton for $\neg\phi$ which accepts just those paths π as input that satisfy $\neg\phi$. The formula is then true just when the language accepted by this automaton intersected with the accepted by the model automaton is empty.

Let ϕ be the LTL property $\mathbf{G}\,(\mathsf{sys.req1} -> F\,\mathsf{sys.stopped\_at1})$.

1. Write $\neg\phi$ in a normalised form, where the negations are pushed inwards so they just surround atomic formulas and the only binary logical connectives used are $\wedge$ and $\vee$. This should simplify the writing of a Büchi automaton for $\neg\phi$.

2. Write a NuSMV module that emulates a Büchi automaton for $\neg\phi$. *Hint:* you should not need an automaton with more than 3 or 4 states, and you might want to make your automaton non-deterministic..

3. Write an LTL property that captures the acceptance condition of the Büchi automaton, that, if true, indicates that there are no accepting runs of the automaton.

Insert your solutions into the file `lift-ltlmc.smv` in the indicated positions at the start. This file include a copy of the modules from `lift.smv`, but with the `main` module renamed to `system` and a new `main` module that composes the system with the negated formula automaton.

4 Submission Instructions

By 4pm on Friday 18th March, please submit your solution NuSMV files with the command

```
submit ar 2 *.smv
```

Please make sure to include your student UUN in each of the files you complete.

Late coursework will be penalised in accordance with the Informatics standard policy. Please consult your coure guide for specific information about this. Also note that, while we encourage students to discuss the practical among themselves, we take plagarism seriously and any such case will be treated appropriately. Please consult your student guide for more information about this matter.

4th March 2016.
(Item 1d in Section 3.2) 12th March 2016.