Formal Verification: Practical Exercise
Model Checking with NuSMV

Jacques Fleuriot Daniel Raggi *

Semester 2, 2017

This is the first non-assessed practical exercise for the Formal Verification course. You will be using the NuSMV model checker to verify properties of a finite state machine model representing a simple telephone exchange.

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 web-page:

http://www.inf.ed.ac.uk/teaching/courses/fv/practical/

For instructions on using NuSMV, see the *Startup Guide* at:

http://www.inf.ed.ac.uk/teaching/courses/fv/nusmv/nusmv-startup.html

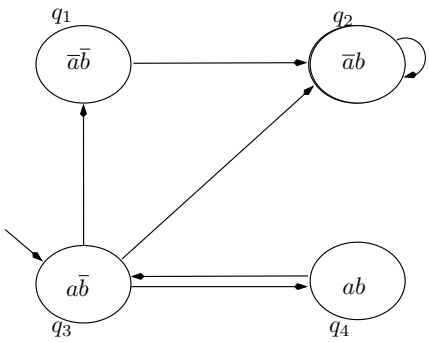

Figure 1: Model for Q1

2 Preliminary Exercises

1. Create a NuSMV model for the system shown in Fig 1. For each of the LTL formulas ϕ below,

   (a) $\mathbf{G}\, a$

   (b) $a\, \mathbf{U}\, b$

   (c) $a\, \mathbf{U}\, \mathbf{X}\, (a \wedge \neg b)$

   (d) $\mathbf{X}\, \neg b \wedge \mathbf{G}\, (\neg a \vee \neg b)$

   (e) $\mathbf{X}\, (a \wedge b) \wedge \mathbf{F}\, (\neg a \wedge \neg b)$

   use NuSMV to (i) determine whether the formula ϕ is valid, and (ii) persuade NuSMV to exhibit some path which satisfies ϕ.

*Originally devised by Bob Atkey.

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. 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.

   (a) $\mathbf{EF}\,\phi$ and $\mathbf{EG}\,\phi$

   (b) $\mathbf{EF}\,\phi \vee \mathbf{EF}\,\psi$ and $\mathbf{EF}\,(\phi \vee \psi)$

   (c) $\mathbf{AF}\,\phi \vee \mathbf{AF}\,\psi$ and $\mathbf{AF}\,(\phi \vee \psi)$

   (d) $\mathbf{AF}\,\neg\phi$ and $\neg\mathbf{EG}\,\phi$

   (e) $\mathbf{EF}\,\neg\phi$ and $\neg\mathbf{AF}\,\phi$

   (f) $\mathbf{A}[\phi_1\ \mathbf{U}\ \mathbf{A}[\phi_2\ \mathbf{U}\ \phi_3]]$ and $\mathbf{A}[\mathbf{A}[\phi_1\ \mathbf{U}\ \phi_2]\ \mathbf{U}\ \phi_3]$.

   (g) $\top$ and $\mathbf{AG}\,\phi \Rightarrow \mathbf{EG}\,\phi$

   (h) $\top$ and $\mathbf{EG}\,\phi \Rightarrow \mathbf{AG}\,\phi$

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

3 Verifying a traffic-light system for a one-lane bridge

3.1 Description of provided traffic-light system model

Figure 2 shows the arrangement of sensors and traffic-lights for a one-lane bridge across a river in Scotland (i.e. drivers are driving on the left).

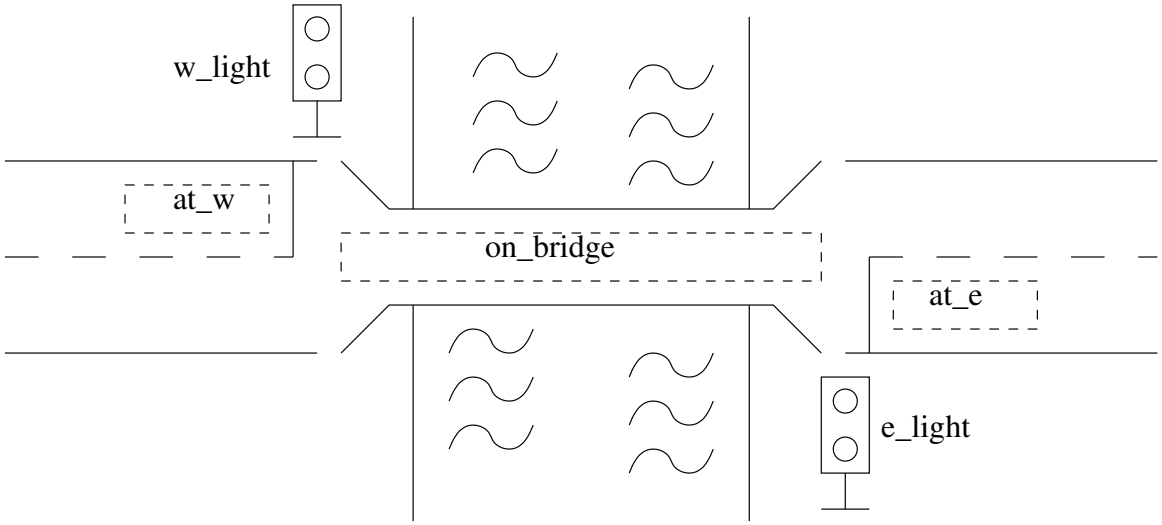

Figure 2: One-lane bridge layout

Built into the road surface in are sensors for detecting whether cars are present in certain regions. These sensors generate boolean-valued inputs to a traffic-light controller. The sensors are

- `at_w`: cars are present at the west end, ready to go onto the bridge,
- `at_e`: cars are present at the east end, ready to go onto the bridge,
- `on_bridge`: cars are present on the bridge.

The traffic-light controller generates outputs with values `rd` (*red*) or `gn` (*green*) that control the traffic light at each end of the bridge:

- `w_light` for the light for traffic coming onto the bridge from the west end,
- `e_light` for the light for traffic coming onto the bridge from the east end.

The provided file `bridge.smv` has a NuSMV model for the traffic light controller. Have a look at this file. The module `main` models the traffic light controller. The internal state of the controller is modelled by the state variables `state` and `tim`. See Figure 3 for a picture of the state machine for the `state` state variable. Implicit in the diagram are self-loop transitions for all conditions not explicitly described. At the bottom of the diagram are the outputs generated by the state machine for each set of states delineated by the dotted lines.

The variable `tim` is an instance of the `timer` module. It counts the number of steps a light stays green for, and ensures a light cannot stay green indefinitely. When `state` has any value but `all_stop1`, the timer runs, and, after a set number of steps, makes a boolean output signal `timeout` true.

3.2 Properties to verify

In the provided template file `bridge-properties.smv`, write formulas for the following properties. Verify your properties with NuSMV by running the command

```
NuSMV -pre cpp bridge.smv
```

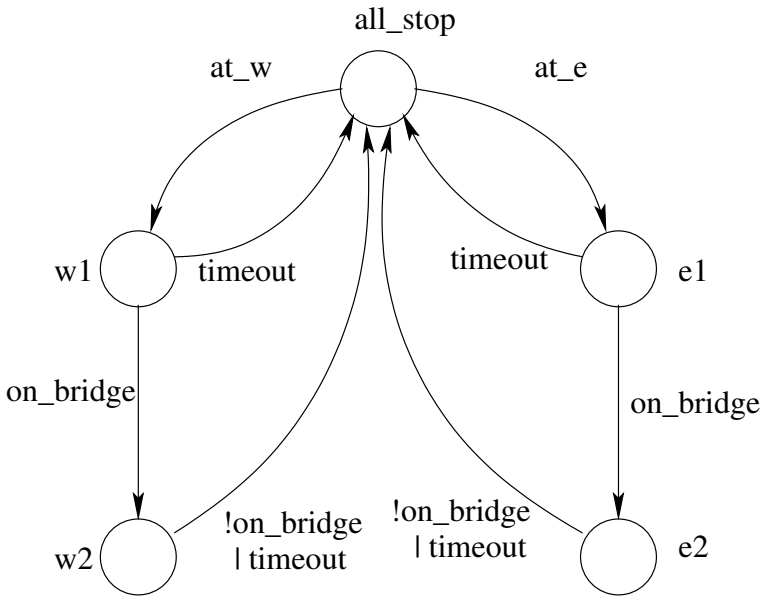

Figure 3: Traffic-light state machine

The `bridge.smv` file brings in the `bridge-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 `bridge.smv` in order to interpret this directive.

If you don't want to see counter-examples for false formulas, also add the `-dcx` option.

1. Write LTL formulas for:

   (a) *It is never the case that both lights are green at same time.* This is an example of a *safety* property.

   (b) *If a light is green, then eventually it goes red.* This is an example of a *liveness* property.

   (c) *The west light is never green for more than 5 consecutive steps.*

   Hint: Think about how to express this property concisely, perhaps first by rephrasing it as some equivalent property.

   (d) *Whenever there is a car at the west light, eventually the west light will go green*, under the assumptions

      i. *if cars are on the bridge, eventually all cars are off the bridge,*

      ii. *if a car is at a red light, it stays at that light up to and including the first step (if there is one) at which the light turns green.*

   Assumption ii is most naturally expressed using the *release* operator (`V` in NuSMV). Express the assumption for cars at the east light using the release operator, and at the west light using the characterisation of weak until in terms of strong until.

   (e) *Whenever there is a car at the west light, eventually the west light will go green*, under the assumption *if cars are on the bridge, eventually all cars are off the bridge.*

   This is the same as the property asked for in part (d), but with the assumption ii about car behaviour omitted. You should find that this property is false. Use bounded model checking to get a short counter-example (see Section 4 below), and write 2 or 3 lines explaining what is going on in this counter example.

When writing NuSMV formulas, note that the precedence of LTL operators (stronger to weaker) is `F G X ! U V & | ->`.

2. Write CTL formulas for:

   (a) There exists a run on the system in which at some point the west light is green and at some point the east light is green.

   (b) Whenever the east light is green and a car is on the bridge, it is possible that the east light then stays green forever

3.3 Controller bug to fix

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

1. In the indicated place in `bridge-properties.smv`, write an LTL property that checks that

   Whenever there is a car at the east light, eventually the east light will go green, under the assumptions

   (a) *if cars are on the bridge, eventually all cars are off the bridge,*

   (b) *if a car is at a red light, it stays at that light up to and including the first step (if there is one) at which the light turns green.*

   This is the same as property 3.2(d), except that it concerns cars waiting at the east light.

   NuSMV should find it false and show a counter-example. Use bounded model checking to find a shortest counter example.

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

2. Make a copy of `bridge.smv` called `bridge-fixed.smv`. Make changes to the light controller code in the `main` module in `bridge-fixed.smv` file to fix this bug. Your changes should address the general problem identified by this bug.

   Do *not* alter `bridge.smv`.

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

3.4 Principles of LTL model checking

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 *if the west light is green, eventually it will go red*, similar to the 2nd LTL property you wrote previously.

As presented in lecture, write a module and LTL specification for $\neg\phi$ that emulates a Büchi automaton for 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.

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

4 Using Bounded Model Checking

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 bridge.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 `bridge-properties.smv` file. If you haven't changed the ordering, this should be the number of the property you write for this question.

5 Help and Feedback

Help with the exercises will be available at the lab sessions in Forrest Hill 1.B19 between 11:10 and 13:00 on Wednesdays. You may also post questions to the class discussion forum at https://piazza.com/ed.ac.uk/spring2017/infr11129/, or contact the TA directly at danielraggi@gmail.com. You **should** discuss your approach and solutions with your fellow students in order to gain peer-feedback.

6 Assessment

This practical exercise is **not** assessed but you should make sure that you attempt it in order to gain practical insight into the applications of model checking. Notes on the solutions will be made available by the end of Part 1 of the course.