# Formal Verification: Practical Exercise[*]
# Model Checking with NuSMV

Jacques Fleuriot  Jake Palmer

Semester 2, 2018

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 First In First Out (FIFO) digital circuit.

## 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 Formal Verification web-page:

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

For instructions on using NuSMV, see the *NuSMV Startup Guide*, linked to from the course page.

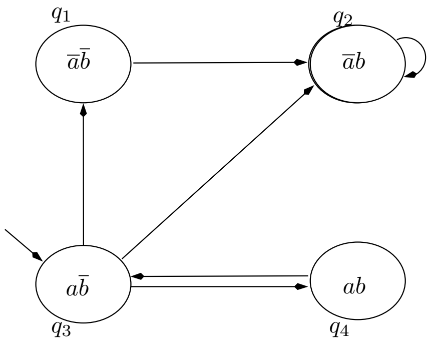

Figure 1: Model for Q1

## 2 Preliminary Exercises

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

[*] Originally devised by Paul Jackson

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

(b) $\mathbf{G}\,(\neg a \Rightarrow \mathbf{X}\, b)$

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

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

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

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

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

*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. In particular, you should understand why both $\phi$ and $\neg\phi$ can be false.

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 FIFO

For this part of the coursework, you verify properties of a model of a FIFO digital circuit.

## 3.1 Description of provided FIFO model

A block diagram of the FIFO (First In First Out) circuit is shown in Figure 2.

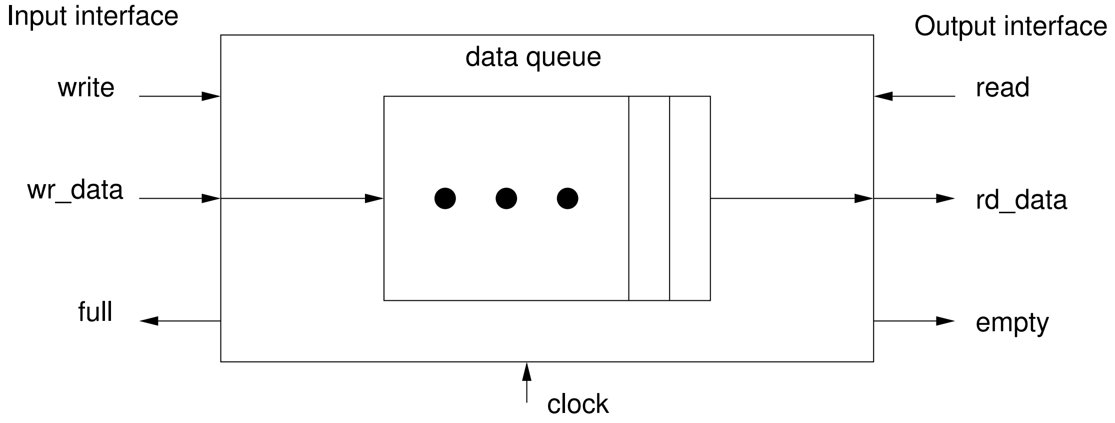

Figure 2: FIFO Block Diagram

Abstractly, a FIFO is a variable-length queue of data words. It has two interfaces, one *input interface* for adding words to one end of the queue and one *output interface* for reading and removing words from the other end of the queue.

The hardware circuit is a *synchronous* circuit. Its behaviour is governed by a Boolean clock signal input which usually alternates between *true* and *false* at a uniform frequency. Each time the clock changes from *false* to *true*, the internal state of the circuit is updated, based on the current internal state and inputs to the circuit at that time. When modelling a synchronous circuit in NuSMV, we do not explicitly include the clock signal. Rather, we design a transition system that takes one step per clock cycle and that uses the transition relation to specify how the internal state is updated based on the current state and inputs. In general synchronous circuits might have outputs that depend both on the current state and the inputs. Here we use a restriction of this scheme where the outputs depend only on the current state. For simplicity, the following description of FIFO behaviour is in terms of the transition system model rather than the hardware circuit.

To add or write a word of data to the FIFO, the data is presented on the *write data* wr_data input and the Boolean signal write is asserted (set to *true*). Providing the FIFO is not currently full, the write data word is then added to the queue on the transition to the next step. The FIFO has a maximum number of words it can hold in the queue at any one time. The Boolean full output of the FIFO indicates whether or not it currently holds the maximum number.

The *read data* rd_data output of the FIFO shows the current end word in the FIFO's internal queue, providing that the queue is not empty. The queue being empty is signalled by the Boolean empty output being set to *true*. If the Boolean read signal is set to *true* and the queue is not empty, on the transition to the next step the current end word in the queue is removed and the word behind it (if any) then appears on the FIFO `rd_data` output.

The provided file `fifo.smv` presents the NuSMV FIFO model. Have a look at the model. For simplicity and to ensure rapid NuSMV execution times, we set the `DEPTH` constant for the maximum number of words to 5 and the `WIDTH` constant for the word size to 1. In practice we would often use larger values for both parameters.

Internally, the FIFO uses a *circular buffer* to implement the queue. This consists of an array `buffer` of words of size `DEPTH` and two pointers into this array, the *read pointer* `rd_p` and the *write pointer* `wr_p`. If the queue is not empty, the read pointer points to word which is the current output word of the queue and, if the queue is not full, the write pointer points to the position to write the next input word. When a new word is written into the queue, the write pointer is incremented, wrapping it around as necessary. When a word in the queue is removed, the read points is incremented, wrapping it around as necessary. See Figure 3 for two examples of the internal configuration of the FIFO when the queue holds the words `w0`, `w1` and `w2`, added in that order.

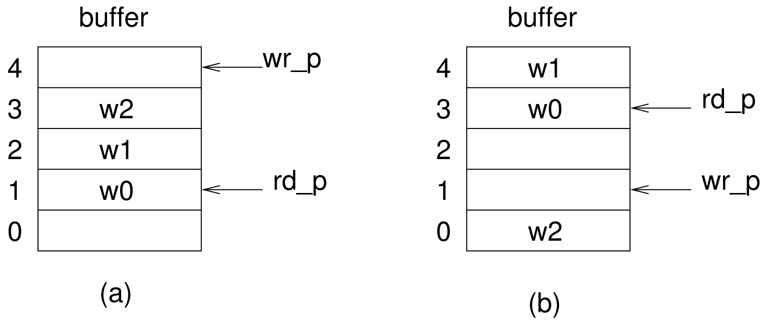

Figure 3: Examples of internal FIFO configurations

With the provided FIFO realisation, the FIFO could be either empty or full when the two pointers are equal. The design uses the Boolean `empty` internal state variable to distinguish between these two cases.

## 3.2 Properties to verify

In the provided template file `fifo-properties.smv`, add formulas for the LTL and CTL properties requested below. Verify your properties with NuSMV by running the command

```
NuSMV -pre cpp fifo.smv
```

The `fifo.smv` file brings in the `fifo-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 `fifo.smv` in order to interpret this directive. You will get warnings until you fill in the LTL specififcations. If you don't want to see counter-examples for false formulas, also add the `-dcx` option.

1. Write LTL formulas for the following properties (all true of the FIFO model in `fifo.smv`):

   (a) the property

> *It is never the case that the FIFO indicates simultaneously it is both empty and full.*

This is an example of a *safety* property. Safety properties in general are about undesired behaviour not happening.

(b) the property

> *If* `write` *is asserted forever and* `read` *is never asserted, then the FIFO eventually becomes full.*

This is an example of a *liveness* property. Liveness properties in general are about desired behaviour actually happening.

(c) the property

> *At any time, if a 1 is presented to the FIFO data input and* `write` *is asserted, then eventually a 1 will appear on the FIFO data output*

with further reasonable assumptions added after the *if* concerning FIFO signals such as `read`, `empty` or `full`, to ensure the property checks true. Consult the NuSMV user guide for information on *word constants* to be able to refer to the number 1.

(d) the same property as in (c), except that it is phrased to hold for any data value, not just the value 1. Use the 'frozen variable' `data1` to do this. Consult the NuSMV user guide for documentation on *frozen variables* (also sometimes known in temporal logic as *rigid variables*).

(e) the same property as in (d), except that, in addition, it requires the `empty` output of the FIFO to be set to false at all times *inbetween* the time the write of the data is set up and the time the data can first be read out, but not actually at either of these times. You may take advantage of the fact that the earliest we expect the data to appear is the step after it is written.

(f) a similar property to that for (d), except that it assumes that two possibly-distinct data values are input on consecutive steps, and checks for the same two values appearing on the output on consecutive steps. Use the provided frozen variables `data1` and `data2` to refer to the two data values.

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 for which, at some time onwards, the FIFO is alway full,*

   (b) *from every reachable state in which the FIFO is full, there exists a path along which the FIFO eventually becomes empty.*

Both of the above properties should be found true of the FIFO model in `fifo.smv`.

## 3.3 Model bug to fix

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

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

   > *always, if the FIFO indicates it is empty, then the read and write pointers are equal.*

   NuSMV should find it false and show a counter-example. Use *bounded model checking* to find a shortest counter example. See Section 4 below for a brief introduction to bounded model checking with NuSMV.

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

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

   Do *not* alter `fifo.smv`.

   At the top of `fifo-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 $\phi$, one constructs a Büchi automaton for $\neg\phi$ which accepts just those paths $\pi$ as input that satisfy $\neg\phi$. The formula is then true just when the language accepted by this automaton intersected with that accepted by the model automaton is empty.

Let $\phi$ be the LTL property $\mathbf{G}\,(\mathsf{full} \land \mathsf{read} \Rightarrow \mathbf{X}\,\neg\mathsf{full})$.

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 $\land$ and $\lor$. 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.

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 solution into the file `fifo-ltlmc.smv` in the indicated positions at the start. This file include a copy of the module from `fifo.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 fifo.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 `fifo-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 **4.12, Appleton Tower** between **11.10 and 13.00** on **Wednesday**. You may also post questions to the class discussion forum at `https://piazza.com/ed.ac.uk/spring2018/fv`, or contact the TA directly at s1673264@sms.ed.ac.uk. 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.