ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A. Lochbihler and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 14: Modelling, LTL, and Model Checking

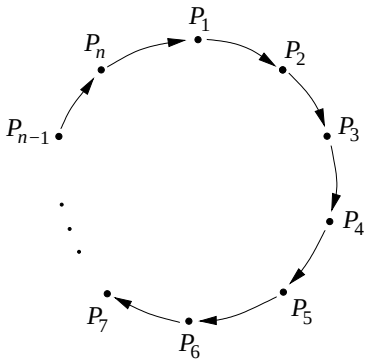
Submission deadline: June 2nd, 2014

Please submit your solution before **11:00am** on the submission date specified above. Solutions can be submitted via e-mail or by using the boxes in front of **CAB F 52.1**. Make sure that the first page (and preferably each sheet) always contains your name, the exercise sheet number and your tutor's name. Don't forget to staple your pages if you submit more than one page.

You can pick up your corrected submissions between **5th June and 20th June** from the boxes in front of **CAB F 52.1**, or you can send a message to your tutor to arrange a meeting.

Assignment 1 (Leader Election)

Consider the following leader election protocol. For $n \geq 1$, the processes $P_1, \ldots, P_n$ are located in a ring topology, where each process is connected by an unidirectional channel to its neighbor as outlined in the figure to the right.



To distinguish the processes, each process has a unique identifier id with $1 \leq id \leq n$. The aim is to elect the process with the highest identifier as the leader within the ring. Therefore, each process executes the following algorithm:

```
send message id
loop
    receive message m
    if m = id then stop
    if m > id then send message m
end loop
```

(a) Model this leader election protocol for n processes in Promela.
Hint: Use an array of n channels of length 1, i.e.,

```
#define N 5  /* number of processes in the ring */
#define L 1  /* length of a channel */
chan c[N] = [L] of { byte };
```

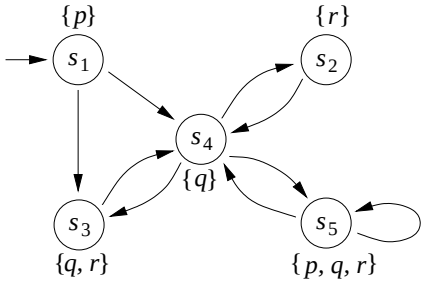
Model a process in Promela as

```
proctype pnode(chan in, out; byte id) {
  /* algorithm for electing the leader ... */
}
```

(b) Assume that the channels are of length $n+1$ instead of length 1 in your Promela model. Is there a state in some execution in which a channel stores more than n messages? Use Spin to verify your claim for some fixed values of n. What happens if the channels have length 0?

Assignment 2 (Linear Temporal Logic)

Consider the transition system T over the set of atomic propositions $P = \{p, q, r\}$:



That is, T is the transition system $(\Gamma, s_1, \rightarrow, L)$ with $\Gamma = \{s_1, s_2, s_3, s_4, s_5\}$,

$$\rightarrow = \{(s_1, s_3), (s_1, s_4), (s_2, s_4), (s_3, s_4), (s_4, s_2), (s_4, s_3), (s_4, s_5), (s_5, s_4), (s_5, s_5)\},$$

and $L(s_1) = \{p\}$, $L(s_2) = \{r\}$, $L(s_3) = \{q, r\}$, $L(s_4) = \{q\}$ and $L(s_5) = \{p, q, r\}$.

(a) Which of the following LTL formulas are satisfied in T, i.e., $T \models \varphi_i$? Justify your answer. If $T \not\models \varphi_i$, provide a computation γ of T such that for the corresponding trace t, $t \not\models \varphi_i$.

$$\begin{aligned}
\varphi_1 &= \Diamond \Box r \\
\varphi_2 &= \Box \Diamond r \\
\varphi_3 &= \bigcirc \neg r \Rightarrow \bigcirc \bigcirc r \\
\varphi_4 &= \Box p \\
\varphi_5 &= p \,\mathsf{U}\, \Box (q \vee r) \\
\varphi_6 &= (\bigcirc \bigcirc q) \,\mathsf{U}\, (q \vee r) \\
\varphi_7 &= \Diamond \Box \bigcirc q \\
\varphi_8 &= (\Diamond \Box p) \Rightarrow (\Diamond \Box r)
\end{aligned}$$

(b) Formalize the following properties in LTL:

(i) Eventually, it will not be possible for the system to go to state s_1.

(ii) Whenever the system is in a state that satisfies r, then the next state satisfies q.

(iii) p always implies r except perhaps in the initial state.

(iv) Whenever the system is in state s_5, it will remain there until r becomes false.

(v) q will be true at least twice.

(vi) q will be true infinitely often.

(vii) If p is true only at the initial state of a trace, then r is false infinitely many times in that trace.

(viii) s_4 can never be repeated (there is no transition from s_4 to itself).

(c) A Promela model for the transition system above is given in file `ts.pml`. Use Spin to model-check all the properties ϕ_i *that do not contain the* $\bigcirc$ *operator*

Note: To model check a Promela file against an LTL formula f with Spin, you may express the LTL formula in an LTL block in the file:

`ltl` $\{f\}$;

The LTL formula is expressed using:

- `[] <> ! && || -> U` for operators $\Box$ $\Diamond$ $\neg$ $\wedge$ $\vee$ $\Rightarrow$ U resp.
- The equality operator as a primitive propositional formula. For example, to express `p`, one should write `(p==true)`.

Spin should be executed (if running from the command-line) with the analysis option `-a`. Furthermore, the final executable must also be executed with the `-a` option. For example:

```
spin -a ts.pml
gcc -o ts.exe pan.c
./ts.exe -a
```

In the guis (e.g. xspin), you can achieve the same thing by choosing (in Verification Parameters) Liveness, acceptance cycles and "Apply Never Claim" /"use claim" (ignore the warnings in xspin - these are because the gui doesn't understand the `ltl` block syntax (but the underlying version of spin does).

(d) Inspect the file `ts.pml` and answer the following question:

- Why are `atomic` statements used?

Assignment 3 (Safety and Liveness Properties)

- Let A be a set of atomic propositions. Prove that a property is both a safety and liveness property, iff it is equal to $\mathcal{P}(A)^\omega$.
- Is $\emptyset$ a safety property, a liveness property, both, or neither?

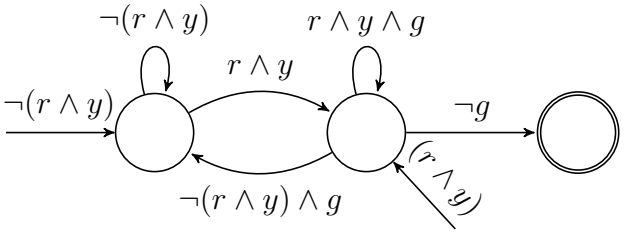
Assignment 4 (Traffic Light)

A "British" traffic light operates as follows. There are three lights: red, yellow, and green. There are four states: red, yellow, green, and "red-yellow". In the latter, both the red and the yellow lights are on. The traffic light performs forever the following transitions: green $\rightarrow$ yellow $\rightarrow$ red $\rightarrow$ "red-yellow" $\rightarrow$ green etc.

The program `ampel.pml` describes precisely the behavior of the British traffic light.

(a) Construct an automaton that describes the behavior of the British traffic light. Use r, y, g as propositional variables to stand for "red light is on", "yellow light is on", "green light is on" resp. Define the value of the propositional variables in terms of the program variables.

(b) The light satisfies the safety property $\Box(r \wedge y \Rightarrow \bigcirc g)$. A *bad-prefix automaton* (describing the bad traces) for this safety property is given below:



(As discussed in your exercise session, an edge labelled with a propositional *formula* (instead of a subset of $\{r, y, l\}$) is a short-hand notation for a number of edges; one for each subset of $\{r, y, l\}$ in which the formula is true. For example, the diagram edge labelled $r \wedge y$ really represents two edges in the automaton: one labelled with $\{r, y\}$ and one labelled with $\{r, y, g\}$).

Construct the product of the automaton describing the program transition system, and the bad-prefix automaton above (you only need to show reachable states!). How does your construction prove the claim that the light satisfies the above property?

Note: To model check the property using the given Promela file, run Spin with the `-DNXT` flag (this must be enabled to support the "next" $\bigcirc$ operator). Unfortunately, (as far as we know) this cannot be achieved from the guis (only on the command-line).

Assignment 5 (Dining Philosophers)

In the previous week, we considered a way to avoid deadlock in the dining philosophers problem. The file `philosophers_no_deadlock.pml` contains the solution that we gave to this problem.

(a) Prove that the solution is *not starvation free*, i.e., there are paths in which a philosopher never eats. You must modify the model accordingly, add an LTL formula, and model-check it with Spin.

(b) Modify the solution to make it starvation free. Model-check the new model with your old LTL formula. You may introduce any restrictions you find appropriate to the behavior of the philosophers.

For this exercise, you may *not* submit handwritten models. Only Promela files will be accepted. While model checking you may use the fairness flag `-f`, but, if you do, be sure to mention it in your solution. As explained in the exercise sessions, the fairness flag causes spin to only explore traces which schedule the individual processes in a *weakly-fair* manner; in particular, it cannot be the case that a process that is continually enabled (could make another step) is never scheduled.

In the guis, this can be chosen by choosing "With Weak Fairness" in the Verification Options.

Note: This exercise is a bit more involved.