Formal Methods in Software Development
Exercise 10 (February 3)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

January 21, 2014

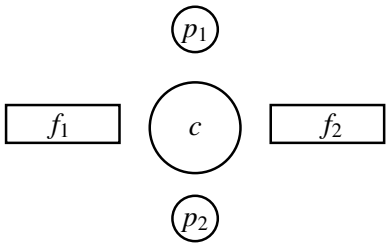
The result is to be submitted by the deadline stated above *via the Moodle interface* of the course as a *.zip or .tgz* file which contains

1. a PDF file with
   - a cover page with the course title, your name, Matrikelnummer, and email address,
   - the deliverables requested in the description of the exercise,
2. the file with the Promela model used in the exercise.
3. the files with the LTL properties (Button "Save As" in the LTL Property Manager).

Exercise 10: Dining Philosophers in Spin

Let us consider a system of two philosophers p_1 and p_2 sitting on the opposites side of a table each with a plate of spaghetti. Each philosopher wants to talk ("philosophize") a bit and then to eat a bit; the philosophers repeat this behavior forever.

The problem is that on the table there are only two forks f_1 and f_2 positioned between the philosophers. In order to eat, a philosopher has to grab one fork and then another fork before she can start eating. However, if both philosophers at the same time grab e.g. their respective left forks, they cannot grab their right forks any more and no one can start to eat (so they all starve).



To overcome the problem, the philosophers place $c = 1$ coins on the table. Before grabbing a fork, a philosopher has to grab a coin. Then she grabs one fork (randomly the left or the right one), then the other fork, and finally starts to eat. When she is done, she returns the forks and the coin.

In general, there are $N \geq 2$ philosophers and N forks and $c = N - 1$ coins; otherwise the protocol remains the same.

The attached file `Philosophers.txt` contains the core of a Promela solution for N philosophers which is suitable for a distributed environment:

- There are N processes `philosopher(`i`)` that represent the philosophers. These processes communicate with the other processes via the following arrays of channels:
  - `crequest` and `creply` represent those channels by which the philosophers communicate with the process `coins()`; both arrays have size N.
  - `frequest` and `freply` represent those channels by which the philosophers communicate with the respective `fork()` processes. Every philosopher has an exclusive pair of channels connecting to her left and to her right fork, i.e., to every fork two pair of channels are connected (thus `frequest` and `freply` have size $2N$).
- There are N processes `fork(`i`)` that manage the forks whose states are represented by the boolean array `f[]` such that `f[`i`]` is true if and only if the fork i is not in use. A fork process may receive from some neighboring philosopher a `GET` request which is answered by a corresponding reply as soon as the fork is available. The philosopher sends a `PUT` message to indicate that it returns the fork.
- The process `coins()` maintains the pool of c coins; it receives from the philosopher processes `GET` requests which are answered by corresponding replies as soon as there is a coin available. The philosopher sends a `PUT` message to indicate that it returns the coin. The process `coins()` selects the message to be processed by a macro `selectRequest(`i`,` c`)` that sets i to the index of a request channel that holds a message that may be accepted in a state in which c coins are available (if $c = 0$, only `PUT` messages are accepted).

Your tasks are as follows;

1. Complete the Promela model by implementing processes `fork(i)` and `philosopher(i)`. A philosopher must non-deterministically choose first her left or her right fork.
2. Run simulations for $N = 2$ and $N = 3$ for several hundred steps. The simulation must not run into a deadlock.
3. Formulate in Spin LTL the property

   > Always, if philosopher 0 is eating, none of its neighbors is eating.

   and model check it for $N = 2$. Analyze the results in detail and explain whether they indicate an error in your model or not.

   Repeat the model check after setting in your model $c = N$ (rather than $c = N - 1$) and explain the results.
4. Formulate in Spin LTL the property

   > At least one philosopher eats infinitely often.

   and model check the property for $N = 2$ (you may specialize the statement for the given value of N). Analyze the results in detail and explain whether they indicate an error in your model or not.

   Select in "Liveness" the option "enforce weak fairness constraint" to ensure that all processes are weakly fairly scheduled; repeat the model check and explain the results.
5. Formulate in Spin LTL the property

   > All philosophers eat infinitely often.

   and model check the property for $N = 2$ (you may specialize the statement for the given value of N) using the "enforce weak fairness constraint" option. Analyze the results in detail and explain whether they indicate an error in your model or not.

Please make sure that the model check truly elaborates the whole state space, i.e., that no message `error: max search depth too small` appears in the output. If such a message should appear, increase in "Advanced Parameters" the "Maximum Search Depth" such that the message goes away (if not, a model check result `error: 0` is meaningless, because only execution paths up to a certain length have been investigated).

Repeat task 4 (both parts) with $N = 3$. Since this model check requires 1 GB memory, it may not run through with the existing memory of your (virtual) machine and the given settting "Physical Memory Available" in "Advanced Parameters"; to solve the problem, you may either increase the amount of memory (in the virtual machine settings and the Spin settings) or reduce the space required by the model checker by setting the "Storage Mode" option "bitstate/supertrace". In the second case you have to check the message that the model checker generates about the `hash factor` it has applied; if this factor is too small (considerably less than 100), hash collisions and consequently not investigated execution paths are likely. If this should be the case, adapt the "Advanced Parameters" option "Estimated State Space Size" accordingly.

The deliverables of the exercise consist of

- The completed Promela model.
- Screenshots of (the final parts of) the simulation runs.
- The LTL properties (PLTL formulas plus definitions of the predicates).
- The output of Spin for each model check.
- Screenshots of counterexample simulations (if any).
- For each model check, an interpretation (did the requested property hold or not and why)?

Some hints/reminders on Promela are given below:

- The Promela version of `if (E) C1 else C2` is

  ```
  if
  :: E -> C1
  :: else -> C2
  fi
  ```

  The Promela version of `if (E) C` is

  ```
  if
  :: E -> C
  :: else -> skip
  fi
  ```

  The Promela version of `while (E) C` is

  ```
  do
  :: E -> C
  :: else -> break
  od
  ```

  In all cases, if you forget the `else` branch, the system will *deadlock* in a state that is not allowed by all the conditions in the other branches.

- The expression *c* `?` `[` *M* `]` is true if and only if a channel *c* holds a message of type *M*. The statement *c* `?` *M* will then remove the message. A typical application is in

  ```
  do/if
  :: cond && c ? [ M ] ->
     c ? M;
     ...
  :: ...
  od/fi;
  ```

  where in a certain situation only a certain kind of message may be accepted.

- In the attached Promela model, the philosophers receive process identifiers 1,3,5,... i.e.

  ```
  philosopher[5]@eating
  ```

  indicates that the third philosopher is at the position of the statement with label `eating` (see the simulations for the identifiers of the individual processes).