Formal Methods in Software Development Exercise 9 (January 24)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

January 12, 2011

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 9: Dining Philosophers in Spin

The attached file `Philosophers.txt` contains the core of a Promela model for the solution of the Dining Philosophers problem modeled in Exercise 8 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()`.
  - `frequest` and `freply` represent those channels by which the philosophers communicate with the `fork()` processes.
- 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 its left or its 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 1 is eating, none of its neighbors is eating.

   and model check it for $N = 2$ and $N = 3$[1].
4. Formulate in Spin LTL the property

   At least one of the philosophers 0 and 1 eats infinitely often.

   and model check the property for $N = 2$ (select in “LTL Verification Options” the option “With Weak Fairness” to ensure that all processes are fairly scheduled, since otherwise the property does definitely not hold).
5. Formulate in Spin LTL the property

   Both philosophers 0 and 1 eat infinitely often.

[1] For $N = 3$ set in “LTL Verification Options” the option “Supertrace/Bitstate” to save space and select in “Set Advanced Options” the maximum search depth to at least 1 million such that the whole search space is traversed.

and model check the property for $N = 2$ (also select in "LTL Verification Options" the option "With Weak Fairness" to ensure that all processes are fairly scheduled).

6. Compare the results of model checking the previous two questions and explain them (in particular why are they the same/different?).
7. Generalize the last two statements for the situation $N = 3$ and repeat the model checks[2].

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)?
- The comparison requested above.

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

[2]Again set the optimization options as indicated above for the case $N = 3$.

```
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).