Formal Methods in Software Development
Exercise 7 (January 14)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

December 6, 2007

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

- A PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - the formal model of the system,
  - the PROMELA code for the system,
  - a screenshot of the "sequence chart" of a sample simulation run.
- a file (.promela) with the PROMELA model of the system.

7: A Manager/Worker System

Consider a system consisting of a single manager process and P worker processes numbered from 0 to $P-1$. The manager maintains an array t of N tasks of type T numbered from 0 to $N-1$. From the task array, the system shall compute a corresponding array r of N results of type R, i.e. it shall reach a state with $r(i) = f(t(i))$, for all task indices i and some computation $f : T \to R$.

The manager also maintains an array w of N numbers in the range $[-1, P]$. If $w(i) = -1$, the computation of task i has not yet started. If $w(i) \in [0, p-1]$, task i is under computation by worker $w(i)$. If $w(i) = P$, task i is completed.

Every worker is in one of the states *idle* or *busy*. When worker number j is *idle*, it may be assigned by the manager some task i with $w(i) = -1$. The task $t(i)$ is then copied into a local variable $l(j)$ of the worker, $w(i)$ is set to j and the status of the worker becomes *busy*. When the worker is finished with the computation, it sets a local "answer" variable $a(j)$ to $f(l(j))$ and returns to state *idle*. When the manager detects that a worker j that has been previously assigned a task i becomes idle, it copies $a(j)$ to $r(i)$ and sets $w(i)$ to P.

1. Develop a formal model S of the system consisting of the state space $State_S$, the initial state condition I_S, and the transition relation R_S.

   Hints: Structure the transition relation by introducing a relation $M_S(\ldots)$ for the transitions of the manager and a relation $W_S(i,\ldots)$ for the transitions of worker i; the predicate $\exists i \in \mathbb{N}_P : W_S(i,\ldots)$ then says "some worker makes a transition".

   An array a of size n with elements from A is represented in the model as usual by a function $a : \mathbb{N}_n \to A$; the expression $\mathbb{N}_n \to A$ denotes the set of all such arrays. Furthermore, as usual, the expression $a[i \mapsto x]$ represents that array which is identical to a except that it holds x at position i.

2. Implement the model in PROMELA: without loss of generality, you may assume that every task is described by a single bit and that the result of the task is the negation of the bit.

   Write the implementation in "message passing style" as shown for the client/server model in class, i.e. use *local* variables for t, r, w, l, a and send messages for the transfers of tasks and results.

   You may define P and N as constants, e.g.

   ```
   #define P 3
   #define N 10
   ```

   To start P processes of type `worker`, you may use the following piece of code in the `init` clause:

   ```
   byte i=0;
   do
     :: (i < P) -> run worker(...); i = i+1;
     :: else -> break;
   od
   ```

   To initialize the elements of array `w` randomly with 0 and 1, write:

   ```
   byte i=0;
   do
     :: (i < N) ->
       if
         :: w[i] = 1;
         :: w[i] = 0;
       fi;
       i = i+1;
     :: else -> break;
   od
   ```

   For more information on PROMELA, lookup the documentation on the web page of Spin.

The result of this exercise consists of the formal model, the PROMELA code, and a screenshot of (a part of) the "sequence chart" of a sample simulation run with three worker processes and ten tasks.