The files

- pollingsystem.prism and
- pollingsystem.props

contain the PRISM model generated for the LPEP example program given in section II.C of the paper

"An Approach to Performance Evaluation Programming"
Author: Eneia Nicolae Todoran (SYNASC 2017 submission 68)

Remark: The paper contains some typos:

- Page 7 bottom (paragraph starting with "We only present two simple…", and the following):
  - s1, s and a are the PRISM counterparts… → vs1, vs and va are the PRISM counterparts…
  - S=? [s1=1 & !(s=1 & a=1)] → S=? [vs1=1 & !(vs=1 & va=1)]
  - Fig III presents → Fig 1 presents
- Page 8 (par. 3):
  - In the experiment given in III we … → In the experiment given in Figure 2 we …

Using the PRISM probabilistic model checker (www.prismmodelchecker.org) we have performed various experiments, including the ones presented in the paper (section III), which we reproduce below.

The following experiment (presented below, and also in Figure 1, section III) shows the probability that in the long run station 1 is awaiting service:

S=? [vs1=1 & !(vs=1 & va=1)]

In this experiment:

- N ranges from 1 to 20 with step 4
- sched ranges from 10 to 50 with step 20

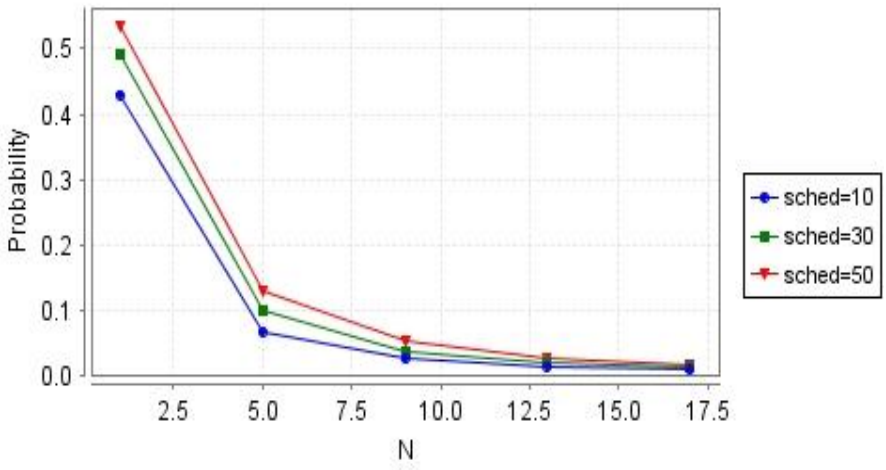

Figure 1. Probability that in the long run station 1 is awaiting service

For the experiment given below (and also in Figure 2, section III) we defined a rewards structure:

```
rewards "served" // expected number of times Station1 is served
        [serve1] true : 1;
endrewards
```

We compute the expected reward (number of times station 1 is served) accumulated by time T as follows:

```
R{"served"}=?[C<=T]
```

In this experiment

- we fixed sched=10
- N ranges from 1 to 20 with step 4
- T ranges from 1 to 15 with step 3

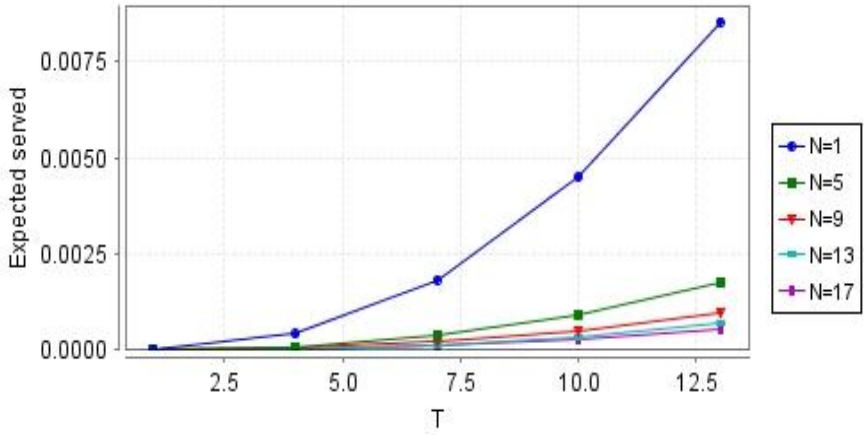

Figure 2. Number of times station 1 is served

Remark: In the file pollingsystem.prism we did not generate PRISM code to monitor the variables vIndex1 (module Station1) and vIndex2 (module Station2), which are used in receive statements ?vIndex1 (rendezvous on action serve1) and ?vIndex2 (rendezvous on action serve2), respectively. The values for vIndex1 and vIndex2 are computed by the function findex in corresponding send activities. For example, assuming that vVal1=1, wfile=cons[Z] 3 (nil[Z]) which is encoded as 4 (see section III), and vmax=3, the send statement (!(((findex vVal1) 1) vmax) wfile) executed by the LPEP command

```
[serve1] (vs=1)&(va=1) -> !(((findex vVal1) 1) vmax) wfile : (va:=2)&(vVal1Cp:=vVal1);
```

yields 2, representing the index of (vVal1=1) in (cons[Z] 3 (cons[Z] 1 (nil[Z]))), which will be the value of wfile after the corresponding call to finsert. The following code is generated in the PRISM file for this command (the function ((((findex vVal1) 1) vmax) wfile) is evaluated in 23 basic or elementary steps):

```
[serve1] (vs=1)&(va=1)&(vVal1=1)&(wfile=4) -> 1/(23 + updsched) : (va'=2)&(vVal1Cp'=vVal1);
```

If we want to monitor the behavior of vIndex1 (and vIndex2) in the PRISM file in module Station1 we could generate the following command (which has a passive rate)

[serve1] (vs1=1) & (vVal1=1) & (wfile=4) -> 1 : (vIndex1'=2) & (vs1'=0) & (vVal1'=vVal1+1);

This value (namely 2) is computed by ((((findex vVal1) 1) vmax) wfile), transmitted to module Station1 and assigned to vIndex1.

In this case we should generate a different receive command (which synchronizes with a corresponding send command executed by module Server) for each different combination of values for vs1 and vVal1, (and similarly for vIndex2, vs2 and vVal2, which are used in synchronizations upon the action serve2).

This option is explored in the (sub) folder furtherExperiments. We can study the behavior of the variables vIndex1 and vIndex2 in the long run but we do not get interesting results, because after a while both vIndex1 and vIndex2 will remain constant. For example, starting from an initial state where wfile= cons[Z] 3 (nil[Z]) (recall that cons[Z] 3 (nil[Z]) is encoded as 4), wfile will evolve as follows:

wfile= cons[Z] 3 (nil[Z])
wfile= cons[Z] 3 (cons[Z] 1 (nil[Z]))
wfile= cons[Z] 3 (cons[Z] 1 (cons[Z] 2 (nil[Z])))
wfile= cons[Z] 3 (cons[Z] 1 (cons[Z] 2 (nil[Z])))
wfile= cons[Z] 3 (cons[Z] 1 (cons[Z] 2 (nil[Z])))
…

Correspondingly, vVal1 will take the following values: 1,2,3,4,4,4,…, and vIndex1 will take the following values: 2,3,1,0,0,0…. We recall that if vVal1 is not found in the list wfile then the server returns constantly 0 (see section II.C in the paper). Hence if we study the behavior of vIndex1 in the long run we get:

S=? [vIndex1=0] = 1
S=? [vIndex1=1] = 0
S=? [vIndex1=2] = 0
S=? [vIndex1=3] = 0

We can get more interesting results if we replace the LPEP command

[serve1] (vs1=1) & (vVal1>=4) -> ?vIndex1' : (vs1:=0) & (vVal1:=vVal1);
with
[serve1] (vs1=1) & (vVal1>=4) -> ?vIndex1' : (vs1:=0) & (vVal1:=1);

In this case vVal1 evolves as follows: 1,2,3,4,1,2,3,4,1,2,3,4,…, and the behavior of vIndex1 in the long run is more interesting. This option is explored in the (sub) folder furtherExperiments.