This folder contains the semantic interpreter for the language $\mathcal{L}_{PEP}$ and the PRISM models described in the paper

"Equivalence Classes in Performance Evaluation Programming"

Author: Eneia Nicolae Todoran (SYNASC 2021 submission 67)

Remark: The initial submission contains some typos; many typos were corrected in current version of the paper (uploaded using EasyChair on September 13, 2021)

- The file `Lpep.hs` contains the semantic interpreter for the language $\mathcal{L}_{PEP}$. The semantic interpreter for $\mathcal{L}_{PEP}$ is implemented in Haskell (`www.haskell.org`). The language $\mathcal{L}_{PEP}$ is described in Section III of the paper.
  - The semantic interpreter can be used to run the $\mathcal{L}_{PEP}$ program presented in Section III.A. The file `Lpep.hs` also contains functions that can be used to generate parameters used in the PRISM models constructed in Section IV.[1]
- PRISM is a widely used probabilitic model checking tool

  `www.prismmodelchecker.org`

- The files `epollingsystem-1.prism` and `epollingsystem-2.prism` contain the PRISM models described in Section IV.A and Section IV.B, respectively.
  - The experiments presented below were performed using the following options (available from the PRISM GUI): "linear equations method" = Gauss-Seidel (Jacobi, the default method does not converge for some experiments), and "Termination max. iterations" = 100000 (the default limit is 10000).
  - The experiments presented in Section IV.B of the paper are described below (based on files `epollingsystem-2.prism` and `epollingsystem-2.props`)
    - Further experiments are provided in the (sub) folder `furtherExperiments` (based on files `epollingsystem-1.prism` and `epollingsystem-1.props`). Due to space limitations, the PRISM experiments provided in the (sub) folder `furtherExperiments` are not included in the paper. The experiments provided in the (sub) folder `furtherExperiments` correspond to the PRISM model described in Section IV.A of the paper.

[1] Further explanations are provided as comments in file `Lpep.hs`

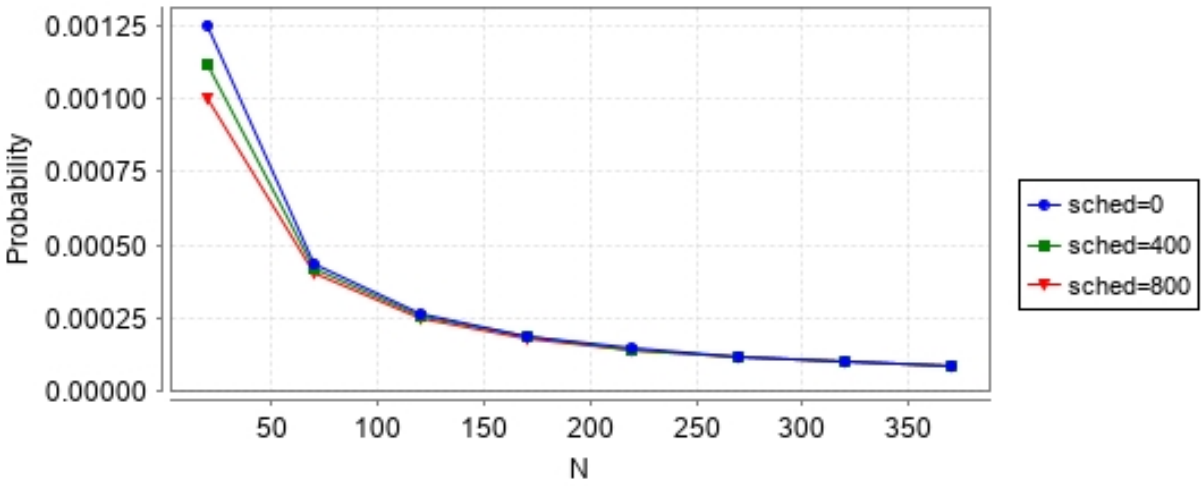

Figure 1: Probability that in the long run station 1 is idle

- The experiments presented in Section IV.B of the paper can be performed by using the PRISM model contained in file `epollingsystem-2.prism` and the properties contained in the file `epollingsystem-2.props`. These experiments are briefly described below (further explanations regarding the meaning of the PRISM variable names employed below are provided in the paper). Section IV.B of the paper considers a partitioning scheme based on abstracting from both the order and the identity of elements stored in variable `wfile`.[2]

  1. Using the PRISM property specification language, the probability that in the long run station 1 is idle can be specified by:

     ```
     S=? [ vs1=0 ]
     ```

     Figure 1 presents a PRISM experiment for this property, where `N` ranges from 20 to 400 with step 50, and `sched` ranges from 0 to 800 with step 400 (`T` is fixed, in this experiment we put `T=10`).

[2]In the PRISM models contained in files `epollingsystem-1.prism` and `epollingsystem-2.prism`, the $\mathcal{L}_{PEP}$ rendezvous interactions between the server and the two stations (synchronizations upon action names a_{serve1} and a_{serve2} in the $\mathcal{L}_{PEP}$ program given in Section III.A) are modeled as synchronizations upon actions `serve1` and `serve2`, respectively, and variables `velem1` and `velem2` (corresponding to variable names v_{elem1} and v_{elem2}) are stored on server. These $\mathcal{L}_{PEP}$ rendezvous interactions could be modeled by storing variables `velem1` and `velem2` on stations, but this would imply the need to generate a larger number of synchronization commands per station. Since PRISM variables can be accessed in read mode from all modules, allocating these variables on server has the same effect in this case.

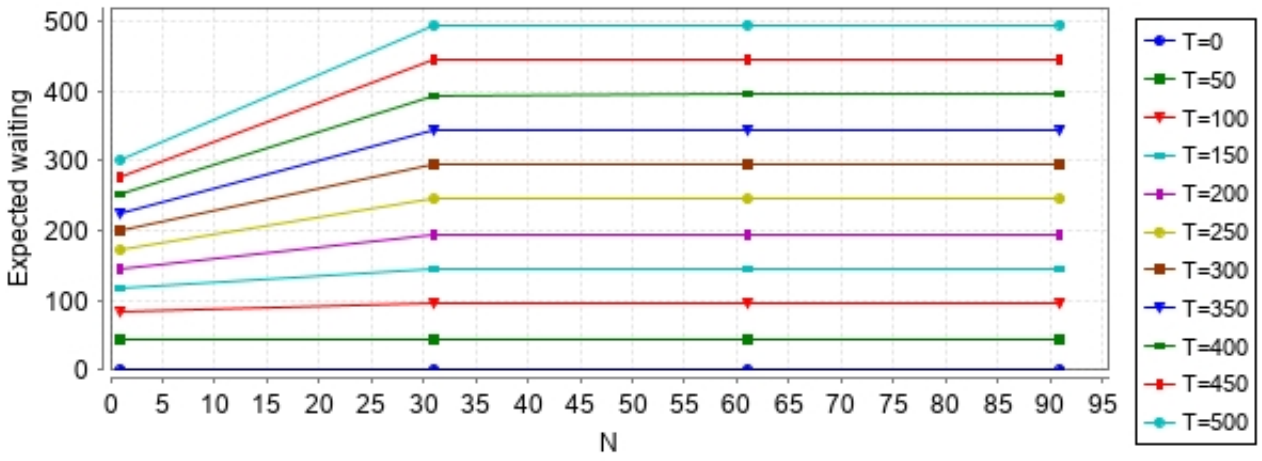

Figure 2: Expected time spent by station 1 awaiting to be served

2. Using the rewards structure "waiting"

```
rewards "waiting"
  vs1>0 & !(vs=1 & va>0) : 1;
endrewards
```

we can compute the expected time that station 1 is waiting to be served accumulated by time T as follows:

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

In the experiment given in Figure 2 we fix the value of `sched` to 50, `N` ranges from 1 to 100 with step 30, and T ranges from 0 to 500 with step 50.