ML1: Protocols and Queueing Delays

The soft deadline for this modelling lab is May 13. The hard deadline is May 17.

Submission: Your submission must be a `.zip` file that contains, at the top-level (i.e. not within any subfolders), the Modest model files with your solutions to the tasks of this lab, using the file names specified in the task descriptions. At the beginning of each file, there must be a multi-line comment (`/* ... */`) containing a brief description of your model and solution, as well as the answers to any particular questions raised in the task description. Document any non-default settings you use for your experiments with modes (except for number of simulation runs), and explain why you need to use them and why you can do so without affecting your results in an undesired way.

Task 1: Questions protocol

Let us return to protocols for asking questions in a lecture. We now study a variant that works as follows:

- Before asking a question, the student must raise their hand and be acknowledged by the lecturer.
- When the lecturer sees the student raising their hand, they immediately decide to either acknowledge or ignore the student. The latter happens with probability $\frac{3}{4}$.
- When acknowledged, the student can ask multiple questions. The student waits for an answer before asking the next question. After getting the answer to the n-th question, the student stops asking questions with probability $\frac{n}{n+1}$.
- The lecturer answers every question. Answering a question takes anywhere between 30 seconds and 2 minutes, uniformly distributed.
- The student can raise their hand to ask a new batch of questions as often as they want. However, this always happens on average 5 minutes after the last answer or the last ignoring, with the actual waiting times following an exponential distribution.
- When the student has been ignored three times in a row, they give up and leave the lecture hall.

(a) Model this protocol in Modest. As a first step, build a behavioural specification that ignores all timing aspects. Use processes to separate the conceptual entities of student and lecturer. Which actions are "initiated" by the student, which actions are "initiated" by the lecturer?

Hints: Use a dedicated action to let the student know when the lecturer decided to ignore them. Count the number of times the student is ignored consecutively. Use `alt` and `when` to write conditions referring to this counter.

(b) Add properties to compute
- the probability that the student eventually leaves the lecture hall,
- the probability that the student leaves the lecture hall before at least 5 questions have been answered, and
- the expected number of questions that have been answered once the student leaves.

Report your results.

(c) Now extend your model with the timing aspects as described above, and add properties to compute
- the expected time until the student leaves the lecture hall, and
- the expected time until either 5 questions have been answered, or the student leaves the lecture hall.

Report your results.

Submit your solution to parts (a) and (b) in a file named `Task1ab.modest`, and your solution to part (c) in a file named `Task1c.modest`.

Task 2: Queueing delays

We want to study a system consisting of a single server with an unbounded queue that behaves as follows: Every `T_ARRIVAL` time units, a request arrives at the server and is stored in the server's queue. Whenever the server is idle and the queue is not empty, the server removes a request from the queue and becomes busy. The server remains busy handling the request for 2 time units. After that, the server becomes idle again.

(a) Model this system in MODEST.

Hints: Use a global integer variable to model the number of requests in the queue. Use the `par` construct to make sure that requests can arrive continuously, independent of the server's current state. You can declare constants using the following syntax:

```
const real T_ARRIVAL = 1.0;
```

If you omit the initialisation to a concrete value, you can provide a value by adding *experiments* like `T_ARRIVAL = 1.0` within mime.

(b) Add five properties to your model that compute the expected length of the queue at times 100, 200, 400, 800 and 1600. Vary `T_ARRIVAL` and use these properties to evaluate the behaviour of the system in terms of queue length for a traffic intensity of 50 %, 80 %, 90 %, 95 %, 100 % and 125 % with modes. Report and describe your results.

(c) Now change the model such that the time between the individual requests is `T_ARRIVAL` *on average* and follows the exponential distribution, and such that the busy times are 2 *on average*, with the individual values being exponentially distributed.

(d) Redo the computations of (b) for the modified model. Use at least 5000 simulation runs for each value of `T_ARRIVAL` to obtain reasonably good results. Report and describe your results. Try to explain why they are so different from the results you obtained in (b).

(e) Now replace the unbounded queue by a queue with a capacity of 10 requests. If a request arrives when the queue is full, it is dropped. Add properties to compute the probability that at least one request is dropped before time 100, 200, 400, 800 and 1600. Add properties to compute the expected number of requests that have been dropped at time 100, 200, 400, 800 and 1600. Vary `T_ARRIVAL` and use these properties to evaluate the behaviour of the system in terms of dropped requests for a traffic intensity of 50 %, 80 %, 90 %, 95 %, 100 % and 125 % with modes. Report and describe your results.

Submit your solution to parts (a) and (b) in a file named `Task2ab.modest`, your solution to parts (c) and (d) in a file named `Task2cd.modest`, and your solution to part (e) in a file named `Task2e.modest`.