Laboratory Assignment 2

Modelling Interference with Finite State Processes (FSP)

Module: Concurrency (CS-210)
Academic year: 2020-21

Allocated marks: This assignment accounts for 2% of the total module marks.

Objectives

The learning objectives of this assignment are as follows.

- To generate a model from a scenario and test potential interference issues.

Resources:

- For this session you will use the LTSA tool. Download the tool using the link on Canvas page: *Modules* > *Resources* > *LTSA Tool*. Unzip the compressed file and you should see a file named *ltsa.jar*; double click on it. The software is fairly intuitive. In case you need a little bit of help with how to use it, please refer to the video tutorial under the `Panopto > Tutorials` section on Canvas.
- There is a cheatsheet on writing FSP in the following link:

  `https://www.doc.ic.ac.uk/~jnm/book/firstbook/ltsa/Appendix-A.html`

  You may find it useful for this session.

Tasks

Consider the following scenario.

> A central computer connected to remote **terminals** via communication links is used to automate **seat** reservations for a **concert hall**. A booking clerk can display the current state of reservations on the terminal screen. To book a seat, a client chooses a free seat and the clerk enters the number of the chosen seat at the terminal and issues a ticket. A system is required which avoids double-booking of the same seat while allowing clients free choice of the available seats.

In this assignment, we will first construct a model of the system that can lead to `ERROR` state.

To begin with, note that, here we can identify *three* important processes: Terminal, Seat, and ConcertHall. We will start modelling the process that is singular in a sense, i.e. does not include other processes. Which one of the above meets this criterion? The seat.

Task 1. Modelling one seat.

Your first task is to model one Seat process.

> Hint: a `Seat` can be `Empty` or `Booked`. If it is not `reserved`, you can `reserve` it. Alternatively, if it is already `reserved` you end up in `ERROR` state. Furthermore, you can `query` the state of `Seat` and it leads you to the same state.

Task 2: Creating a process for multiple seats.

Your task is to create a composite process `Seats` from two `Seat` processes, with the seats labelled as `0` and `1`.

> Hint: A way to label multiple processes is the following `{a,b}:P`. In this case, we create two process `a:P` and `b:P`, and all actions are prefixed with `a` and `b`. However, here we have numerical labels. To use that, we can use a `range IDs = 0..1` to label the generated processes. Think, what would could `a[IDs]` achieve?

Task 3.Create a `Terminal` process.

Your next task is to model a `Terminal` that allows the user to `query` then `reserve` if `Empty`, or `return` to `Terminal` state.

> Hint: you may find the `label[index:0..N]` syntax useful here.

Task 4: Create the composed process.

You last modelling task is to compose two `Terminal` processes with labels `a` and `b`, when they are sharing your `Seats` process.

Task 5: Answer the following question.

What happens if you do *check* → *Supertrace*? Do you see an error? If so, why?

Once you have completed all the tasks, please make sure that you have been signed off.