The University of Melbourne
School of Computing and Information Systems
SWEN90004 Modelling Complex Software Systems

Assignment 1b, 2017

Released: Wednesday 15 March. Deadline: 17:00, Thursday 13 April.

Objective

To use a higher-level modelling language to specify and reason about a concurrent system.

Background and context

Assignment 1 has two parts. The first part, 1a, was worth 10% of your final mark; this part, 1b, is worth 15%. In the first part (which you should complete before attempting this part) you designed and implemented (in Java) a simulator for a bicycle quality control system. In fact, two solutions were asked for: a solution that (typically) failed to ensure progress of all components, and a second solution that avoided that problem. Now the task is to model both systems in FSP, to use LTSA to check both, and to identify any problems that may be discovered through modelling.

Note: for the purpose Assignment 1b, you should assume that there are **no** defective bicycles. That is, you may assume that the inspector removes the tags from **all** bicycles after inspection. Thus, **all** bicycles will leave the inspector without a tag.
This should not impact concurrent aspects of system behaviour, but will reduce the size of your model slightly, as you can ignore the `defective` property of bicycles and don't need to worry about bicycles being returned to the belt whilst still being tagged (as this will not occur). You can simulate this scenario in your Java implementation by setting `Params.DEFECT_PROB` equal to 0.

The tasks

There are three documents to submit and five sub-tasks to do. Remember to include your name on each submitted document.

First, model and check the two solutions you submitted in Assignment 1a. For each:

- **Model:** Model your Java implementation in FSP. That is, reverse engineer a FSP model from the Java code. Try to capture the essential interactions between the components—no more than that. The models should contain comments that explain the design and its components.

- **Check:** Specify what you believe are the relevant safety and liveness properties for your FSP model. Use LTSA to check these properties.

When these four sub-tasks have been completed, reflect on the experience, keeping your discussion to 400 words or less:

- **Discussion:** Which problems did you find in the two models as a result of using LTSA? Were these problems also present in the Java implementation? If you find problems with the Java solution submitted for part 1a, Solution 2, why do you think you picked these up now and not before submitting the Java code? If you did not find problems with your part 1a, Solution 2 submission, were you convinced when you submitted part 1a, Solution 2 that no problems existed? Why did you believe this? Do you still believe there are no problems?

Procedure and assessment

The assignment should be completed by students individually. The submission deadline is Thursday 13 April at 17:00. A late submission will attract a penalty of 2 marks for every calendar day it is late. If you have a reason that you require an extension, email Nic *well before the due date* to discuss this.

To tackle the assignment, first work through (and *understand*) the examples from lectures, and do the workshop exercises. FSP is not difficult—it is simpler than most programming languages, and much simpler than languages like Java. However, as with other languages, the way to master it is to use it, and to learn by doing. Trying to do the assignment straight up means you may struggle. Work through some easier examples first.

Submit a single zip file via the LMS. The file should include

- A file called `sol1.lts` with an FSP model for part 1a, Solution 1, including the relevant safety and liveness properties.
- A file called `sol2.lts` with an FSP model for part 1a, Solution 2, including the relevant safety and liveness properties.
- A file called `discussion.txt`, containing the discussion of issues.
- Optionally, and only where relevant, a directory containing an updated (corrected) Java implementation for part 1a, Solution 2, in case LTSA discovered a problem.

We encourage the use of the LMS's discussion board for discussions about the project. However, all submitted work is to be your own individual work.

This project counts for 15 of the 50 marks allocated to project work in this subject. Marks will be awarded according to the following guidelines:

Criterion	**Description**	**Marks**
Clarity, abstraction	Descriptions of all actions and processes are clear and concise. FSP models are at a suitable level of abstraction. Only the behaviour relevant to interaction is specified, and there is sufficient detail to implement the concurrency objectives from the model.	4 marks
Correctness	Both models accurately reflect the original implementations. The Solution 2 model behaves correctly, is free from deadlock, does not violate any safety properties, and demonstrates liveness properties.	4 marks
Completeness	The model is complete. All components have been modelled and all expected behaviour is present. All expected safety and liveness properties have been described.	3 marks
Formatting	The FSP source adheres to the code format rules from Part 2a where this makes sense, including the use of comments.	2 marks
Discussion	The discussion shows understanding of the subject material.	2 marks
Total		15 marks

Why backwards?

A valid question: Why are we modelling the quality control system *after* implementing it? Should it not be done the other way? Well, yes and no. Many people use modelling to understand an existing code base (just look at the number of tools for reverse engineering UML models from code bases). Reverse engineering is a great way to understand problems with an existing system; for example, why a deadlock is occurring. It is true, however, that in most cases, it would be cheaper and easier to do the modelling first.

The other reason why the assignment is “backwards” is that trying to model a new system using a new type of notation, such as FSP, will often end in disaster. We hope that, having gone through the Java programming stage, you feel familiar with the system to be modelled and thus can concentrate on the use of FSP. The exercise should be one of applying abstraction—a skill that is of utmost importance in any engineering discipline.