

6SENG006W Concurrent Programming

Week 02 Tutorial: FSP using LTSA & Role Playing Concurrency Problems

Introduction

The aim of these tutorial exercises is to introduce:

- Some of the ideas & concepts of concurrent systems, via role playing two classic concurrency problems.
- The FSP language & LTSA tool. We will do this by constructing, animating & analysing simple FSP processes using the LTSA tools.

Exercise 1.1

Form groups of 5 people & role play the Dining Philosophers problem.

If you can not get 5 people then do it with at least 3 people.

Investigate the problem using different scenarios & attempt to identify the problems that can arise & how they could be solved.

Makes brief notes of what you find out.

Exercise 1.2

Form groups of 5 people & role play the Student bank account system. 4 play the individuals & 1 plays the role of the bank.

If you can not get 5 people then do it with at least 3 people.

Investigate the problem using different scenarios & attempt to identify the problems that can arise & how they could be solved.

Makes brief notes of what you find out.

Exercise 1.3

Read the information about the:

- Finite State Process (FSP) language
- LTSA Tool
- FSP Analysis & Design Forms

in the *FSP Information* folder.

Then download the LTSA tool (`ltsa.jar`) & run it.

Exercise 1.4

Type each of the following processes into a **separate** plain text file with the suffix ".lts", e.g. `deadlock.lts`, `rhyme.lts`, etc.

1. The process that represents a deadlocked process - `STOP`.

```
DEADLOCK = STOP.
```

What is the behaviour of this process? What actions does it offer us?

2. The RHYME process:

```
RHYME = ( one -> two -> buckle -> my -> shoe -> STOP ).
```

3. The DRINKS process from Lecture 1.

4. The COINTOSS process:

```
COINTOSS = ( toss -> ( head -> COINTOSS  
                  | tail -> COINTOSS ) ).
```

5. The CHANGE process:

```
CHANGE = ( fivep -> ( onep -> onep -> onep -> onep -> onep -> CHANGE  
                  | twop -> twop -> onep -> CHANGE ) ).
```

Load each FSP process into the LTSA tool, then analyse & animate it.

Exercise 1.5

For each of the FSP processes in **Exercise 1.4** write down its process "*attributes*", these can be found from the LTS tool:


- *alphabet*: the set of actions that a process can perform.
- *transitions*: the transitions between a processes states.
- a possible *trace*: a sequence of actions it can perform.
- if it can "*deadlock*"; unable to do any further actions, i.e. stuck.
- if it can deadlock the *trace* that results in deadlock.

Exercise 1.6

For each of the following processes Labeled Transition System (LTS) graphs (see below for diagrams), give the Finite State Process (FSP) program that would produce it.

Confirm that your FSP process descriptions are correct by using the LTSA tool to generate the LTS diagram for **your process** & compare them.

1. MEETING
2. JOB
3. GAME

In the current version of LTSA the "two one" actions will be displayed as "{ two, one }" or "{ one, two }". But you may need to use the "Minimize" command on the tool bar - ; one of the things this does is to combine equivalent states.

4. MOVE
5. FORTICK
6. PERSON

Similar issues as with GAME.

Exercise 1.7

Once you have defined the FSP processes for each of the LTS graphs from **Exercise 1.6**, write down the process "*attributes*" as listed in **Exercise 1.5** for each of them.

Exercise 1.8

Are the FSP processes you have defined for each of the LTS graphs from **Exercise 1.7** the only ones that can produce the LTS graphs?

Which ones have alternative definitions & which do not?

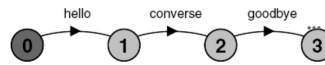
Can you give at least one of the alternative process definitions for those that have?



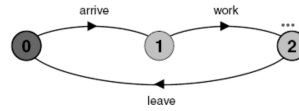
Last updated: 2/10/22

Exercise 1.6 Diagrams

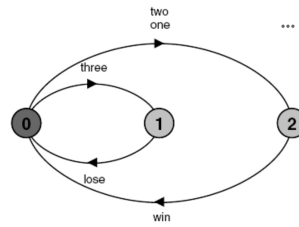
1. MEETING



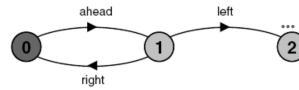
2. JOB



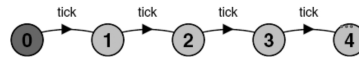
3. GAME



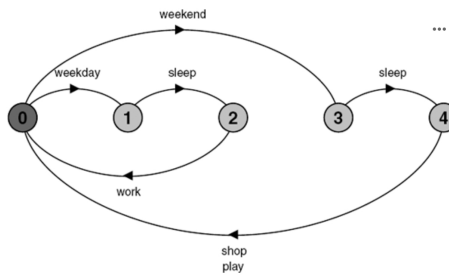
4. MOVE



5. FORTICK



6. PERSON



Solution will be available before the next tutorials.

