COMP30112: Concurrency Exercises

Howard Barringer

February 2008

Some of these exercises are taken from Magee and Kramer's book 'Concurrency', which contains further exercises for you to attempt. Some of the other exercises were prepared by Mark Jacobson, an ex-student in the School.

Important: Clearly, several of the answers to the exercises below can be found by utilising the available tools (LTSA, Java). However, you should do these exercises for the moment just on paper.

Try out your own examples: As we progress with the course, please do not constrain yourself to the given exercises and examples alone — try out your own modelling, analysis and implementation examples, e.g.

- Does your third year project involve concurrency? If so, try building abstract models in FSP.
- Have you got your Threads tangled in the past? If you have encountered problems then try reconstructing the example and analyse it
- Other course units
- Applications

Topic 2.1: Basic FSP Processes

1. Make sure you have drawn the three 'DAY' LTSs, representing the actions of someone getting up and going to work:
   (a) **DAY1**: get up (action `up`), then have tea (action `tea`), then go to work (action `work`), then stop
   (b) **DAY2**: do DAY1 repeatedly
   (c) **DAY3**: do DAY2, but choose between `tea` and `coffee`
2. Write the FSP process definitions for the above. Later, you can check these using the LTSA tool.
3. Extend DAY3 to include the effects of an alarm with a snooze button, so prior to the `up` action, an `alarm` action is performed. However instead of then doing `up` you may do a `snooze` action and go back to the start.

   Answer:

```
DAY4 = (alarm -> (snooze -> DAY4
                 |up -> (tea -> REST
                        |coffee -> REST))),
REST = (work -> DAY4).
```

For the following, write an FSP specification and draw the corresponding LTS diagram. Check manually that your FSP and LTS correspond. Again, later on, you can use LTSA tool to confirm this (LTSA may also be used to animate the specifications)

4. A variable stores values in the range $0..N$ and supports the actions read and write. Model the variable as a process, VARIABLE, using FSP.

   Answer:

   ```
   const N = 2

   VARIABLE = (read[i:0..N] -> VARIABLE[i]),
   VARIABLE[i:0..N] = (read[j:0..N] -> VARIABLE[j]
                      |write[i] -> VARIABLE[i]).
   ```

   For $N = 2$, check that it can perform the actions given by the trace:

   write.2->read.2->read.2->write.1->write.0->read.0

5. A bistable digital circuit receives a sequence of trigger inputs and alternately outputs 0 and 1. Model the process BISTABLE using FSP, and check that it produces the required output; i.e. it should perform the actions given by the trace:

   trigger->1->trigger->0->trigger->1->trigger->0 $\cdots$

   (Hint: The alphabet of BISTABLE is $\{[0], [1], \mathsf{trigger}\}$.)

   Answer:

   ```
   BISTABLE = ONE,
   ZERO = (trigger -> [0] -> ONE),
   ONE  = (trigger -> [1] -> ZERO).

   menu RUN  = {trigger}
   ```

6. A sensor measures the water level of a tank. The level (initially 5) is measured in units 0..9. The sensor outputs a low signal if the level is less than 2, a high signal if the level is greater than 8 and otherwise it outputs normal. Model the sensor as an FSP process, SENSOR.

   Answer:

   Hint: The alphabet of SENSOR is $\{\mathsf{level}[0..9], \mathsf{high}, \mathsf{low}, \mathsf{normal}\}$. When the sensor receives a new level it should output low, normal or high as required. This can be done either via a choice, or by specifying that each level input is followed by the appropriate output.

7. A drinks dispensing machine charges 15p for a can of Sugarola. The machine accepts coins with denominations 5p, 10p and 20p and gives change. Model the machine as an FSP process, DRINKS.

   Answer:

   Hint: It is slightly awkward to specify the set of coins or values in FSP notation — each one is essentially a label index, i.e. `set Coins = {[5], [10], [20]}`. The drinks machine should accept coins until it reaches 15 and then give a drink and coins as change:

   ```
   ...
   CREDIT[5] = (in.coin[5]   -> CREDIT[10]
   ...
   CHANGE[5]  = (can -> out.coin[5]  -> DRINKS),
   ...
   ```

8. A miniature portable FM radio has three controls. An on/off switch turns the device on and off. Tuning is controlled by two buttons scan and reset which operate as follows. When the radio is turned on or reset is pressed, the radio is tuned to the top frequency of the FM band (108 MHz). When scan is pressed, the radio scans towards the bottom of the band (88MHz). It stops scanning when it locks on to a station or it reaches the bottom (end). If the radio is currently tuned to a station and scan is pressed then it starts to scan from the frequency of that station towards the bottom. Similarily, when reset is pressed the receiver tunes to the top. Using the alphabet $\{\mathsf{on}, \mathsf{off}, \mathsf{scan}, \mathsf{reset}, \mathsf{lock}, \mathsf{end}\}$, model the FM radio as an FSP process, RADIO.

   Answer:

   Hint: Use a separate thread to change the frequency. Its `run()` method will react to changes in the state of the radio — whether it's on/off, whether a station has been found and it is tuned etc. Buttons for (on,off,scan, reset) will control the state. Use integer values for frequencies and some simple way of determining if it is tuned (e.g. if the frequency is divisible by 4). Do the state changes need to be synchronised?