Communication and Concurrency
Exercise Sheet 2

The deadline for this coursework is 4.00pm on Monday 18th November. Please submit your solutions to the ITO. The answers should be submitted to the ITO where the answer to Q3 should be a text file you have run on the CWB.

1. Consider the following process definitions. *[20 marks]*

$$\begin{array}{lll@{\qquad}lll}
A_0 & \stackrel{\text{def}}{=} & a.A_1 + \tau.A_0 & B_0 & \stackrel{\text{def}}{=} & a.B_0 + \tau.(B_0 + B_1) \\
A_1 & \stackrel{\text{def}}{=} & b.A_1 + \tau.A_2 & B_1 & \stackrel{\text{def}}{=} & b.B_2 + \tau.B_1 \\
A_2 & \stackrel{\text{def}}{=} & \tau.A_1 + b.A_2 & B_2 & \stackrel{\text{def}}{=} & a.B_0 + \tau.(B_2 + B_1)
\end{array}$$

$$\begin{array}{lll@{\qquad}lll}
C_0 & \stackrel{\text{def}}{=} & a.C_0 + \tau.C_1 & D_0 & \stackrel{\text{def}}{=} & a.D_0 + \tau.D_1 \\
C_1 & \stackrel{\text{def}}{=} & b.C_1 + a.C_0 & D_1 & \stackrel{\text{def}}{=} & b.D_1 + \tau.D_2 \\
 & & & D_2 & \stackrel{\text{def}}{=} & a.D_0 + \tau.D_1
\end{array}$$

(a) Draw the transition graphs of the processes (with respect to 'thin" transitions $\xrightarrow{a}$).

(b) Which of the (different) pairs of processes A_0, B_0, C_0 and D_0 are weakly bisimulation equivalent? Construct a weak bisimulation which contains the processes and prove that it is a weak bisimulation when equivalent and provide an explanation when not equivalent.

(c) Design a process E_0 such that

$$(A_0 \mid E_0) \quad \sim \quad (B_0 \mid E_0) \quad \sim \quad (C_0 \mid E_0) \quad \sim \quad (D_0 \mid E_0).$$

2. Pedestrian crossings are designed to keep the traffic stopped for as long as it takes the pedestrian to cross, rather than just for a fixed time. Such crossings work by having a sonar sensor which repeatedly 'pings' the crossing space to detect objects in it. *[10 marks]*

Consider a well-behaved pedestrian using such a crossing, who may be modelled as follows:

$$\begin{array}{rcl}
\texttt{GoodPedestrian} & \stackrel{\text{def}}{=} & \overline{\texttt{req}}.\texttt{AwaitingSignal} + \texttt{noping}.\texttt{GoodPedestrian} \\
\texttt{AwaitingSignal} & \stackrel{\text{def}}{=} & \texttt{green}.\texttt{InCrossing} + \texttt{noping}.\texttt{AwaitingSignal} \\
\texttt{InCrossing} & \stackrel{\text{def}}{=} & \texttt{noping}.\texttt{Safe} + \texttt{ping}.\texttt{InCrossing} + \texttt{red}.\texttt{Litigant} \\
\texttt{Safe} & \stackrel{\text{def}}{=} & \texttt{noping}.\texttt{Safe} + \texttt{red}.0 \\
\texttt{Litigant} & \stackrel{\text{def}}{=} & \overline{\texttt{sue}}.0
\end{array}$$

In this model $\overline{\texttt{req}}$ is the pedestrian's request to cross; `green` (`red`) is the pedestrian observing the green man (red man); the action `ping` models the observation by the sensor that the pedestrian is in the crossing (the sonar 'ping' is reflected back to the sonar), and `noping` models the observation by the sensor that the pedestrian is not in the crossing (the sonar 'ping' is not reflected); and the action $\overline{\texttt{sue}}$ indicates that the pedestrian is going to sue you for mental distress, as the red man appeared while they were still on the crossing.

(a) Write a CCS definition of a `Controller` process which responds to the pedestrian's request, and the sonar pings, and controls the lights to achieve the informal specification at the start of the question. (You may assume that only one pedestrian at a time is interacting with the crossing.)

(b) Consider the process $\texttt{System} \equiv (\texttt{GoodPedestrian} \mid \texttt{Controller})\backslash L$, where L is all actions except $\overline{\texttt{sue}}$. If you have correctly implemented the controller, it should be impossible for the pedestrian to sue you, and hence you should have

$$\texttt{System} \approx \texttt{0}.$$

Prove this.

3. Your answer to this question should be in a text file that you have checked with the CWB. You are given a chain of five Christmas lights connected by a cable: *[20 marks]*

$$1 - - - 2 - - - 3 - - - 4 - - - 5$$

Each light has a small programmable controller which can send a signal to the light, making it flash. The controller can also send signals to its neighbour controllers on its right and on its left. The exercise consists of programming the controllers in such a way that the lights flash in the following order: $2, 5, 3, 1, 4, 2, 5, 3, 1, 4, \ldots$.

We now translate this informal specification into a more formal one. You are requested to design a process

$$\texttt{Light} \equiv (\texttt{Controller}_1 \mid \ldots \mid \texttt{Controller}_5)\backslash L$$

satisfying the following properties:

0. `Light` can only execute the actions $\texttt{flash}_1, \ldots, \texttt{flash}_5$, and τ.
1. For $i = 1, \ldots, 5$, $\texttt{Controller}_\texttt{i}$ can execute an action $\texttt{flash}_\texttt{i}$, but none of the actions $\texttt{flash}_\texttt{j}$ for $j \neq i$.
2. $\texttt{Controller}_1$ can only communicate with $\texttt{Controller}_2$.
3. For $i = 2, 3, 4$, $\texttt{Controller}_\texttt{i}$ can only communicate with $\texttt{Controller}_{\texttt{i}-1}$ and $\texttt{Controller}_{\texttt{i}+1}$.
4. $\texttt{Controller}_5$ can only communicate with $\texttt{Controller}_4$.

5. `Light` is weakly bisimilar to the process

$$\texttt{Flash} \overset{\mathrm{def}}{=} \texttt{flash}_2.\texttt{flash}_5.\texttt{flash}_3.\texttt{flash}_1.\texttt{flash}_4.\texttt{Flash}$$

The Workbench command `sort P` yields the set of actions appearing in the syntactic description of the process `P`. The conditions 0-4 above can be rephrased in terms of `sort`. For instance, properties 1 and 2 are rephrased as follows:

1. For $i = 1, \dots, 5$, the result of the command `sort` $\texttt{Controller}_\texttt{i}$ contains the action $\texttt{flash}_\texttt{i}$, but none of the actions $\texttt{flash}_\texttt{j}$ for $j \neq i$.
2. If $i \neq 2$, then the commands `sort` $\texttt{Controller}_1$ and `sort` $\texttt{Controller}_\texttt{i}$ must yield disjoint sets of action names.

The Workbench command `eq(P, Q)` checks if the processes `P` and `Q` are weakly bisimilar. So property 5 can be rephrased as follows:

5. The result of `eq(Light, Flash)` must be true.