Concurrency TT 2018: Practical 1

Julian Gutierrez

with thanks to T. Gibson-Robinson, M. Goldsmith, G. Lowe, B. Roscoe

Introduction

The purpose of this practical is to get you familiar with the syntax of CSP_M and FDR. Your solution to this practical should consist of commented CSP_M script that includes answers to the questions below. You can comment in a CSP_M file as follows:

```
-- This is a single line comment
{-
This is a
multiline comment.
-}
```

This practical does not count towards your overall assessment mark, but you must present the results to a demonstrator before proceeding with the next assignment.

Deadline: the beginning of your second practical session (Week 2).

Reading: You should read Section 3.2 of *Understanding Concurrent Systems* before doing Part 2. You may wish to consult the FDR manual, available at `https://www.cs.ox.ac.uk/projects/fdr/manual/`, during the practicals.

Part 1: Introducing CSP_M and FDR

Copy the CSP script from Figure 1 and save the file as `example1.csp`.

The CSP processes in that file are very simple ones which input and output values of type `T` on the channels `input` and `output` respectively.

1. Run FDR on this file and use Probe to make sure that you understand how the processes work. FDR can be launched by either double clicking on the file, or by typing:

   ```
   > fdr3 example1.csp
   ```

   into a command prompt. Once FDR launches, Probe can be launched on `Proc1`, for example, by typing:

   ```
   > :probe Proc1
   ```

```
-- Defines T as the set {1,2,3,4,5}
T = {1..5}

-- Declares the events input.1, output.1, input.2 etc.
channel input, output : T

Proc1 = input?x -> output.x -> Proc1

Proc2 = input?x -> Proc2'({x})

Proc2'(X) =
  output?a:X -> Proc2'(diff(X,{a}))
  [] input?b -> Proc2'(union(X,{b}))
```

Figure 1: File `example1.csp`

into the FDR command prompt.

Hint: The FDR command prompt is a full interactive evaluator for $\mathrm{CSP}_{\mathrm{M}}$. For example, try typing `100/5`, or `member(3, T)`.

2. What is the difference between `Proc1` and `Proc2`?

3. We know that a process P is trace refined by another process Q (or, equivalently Q refines P), written:

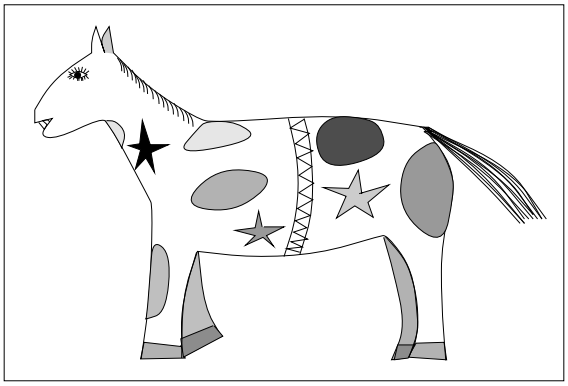
$$P \sqsubseteq_T Q \text{ iff } \mathit{traces}(P) \supseteq \mathit{traces}(Q).$$

   Does `Proc2` refine `Proc1`? Does `Proc1` refine `Proc2`? Explain your answers.

4. Check your answers to question 3 by running both (traces) refinement checks through FDR. If a refinement fails, then check the counterexample found and make sure you understand both how to interpret the counterexample, and why it occured.

   Hint: In order to get FDR to check if $\mathit{Spec} \sqsubseteq_T \mathit{Impl}$, then add a line of the form `assert Spec [T= Impl` to `example1.csp`. Then, reload the file in FDR by typing `:reload` at the FDR command prompt. The assertion(s) should now be listed on the right-hand side of the main window and can be run by clicking Run.

   If an assertion fails, the counterexample can be viewed by clicking Debug. As this is a trace assertion, any counterexample will be of the form `<x, y, z>` where the specification (i.e. the process on the left) cannot perform the event `z` after the trace `<x, y>`. The debug viewer illustrates this by showing the behaviour of the implementation (i.e. the process on the right) below the behaviour of the specification, and drawing the final event in red (to indicate an error). Try clicking on a circular node and see what information is revealed.



5. Currently, `Proc2` will allow an infinite number of `input.x` events to occur consecutively. Define a process `Proc3` that behaves like `Proc2`, except that once `Proc3` has input a value, it cannot input that value again until it has output it; for example, it cannot perform the trace `<input.1, input.1>`, but can perform the trace `<input.1, output.1, input.1>`. Use should use ProBE to check that `Proc3` is functioning correctly.

   Use FDR to check that the expected refinement relationships hold between `Proc3` and `Proc1` and `Proc2`, respectively.

   Hint: In CSP_M `diff(X, Y)` computes $X \setminus Y$. It is also legal to write `c?x:diff(X,Y)`, which will only allow events of the form `c.x` where `x` is in `diff(X, Y)`.

Part 2: Pantomime Horse

This example is taken from *Understanding Concurrent Systems* (page 49) and tutorial exercises. The aim is for you to write the CSP_M script based on the processes presented below and perform the corresponding refinement checks using FDR.

A pantomime horse has two actors in it: one playing the front and one playing the back. You can think of it as a parallel composition of two processes: the two halves have to co-operate on moving, but each has control of some other activities:

$$Horse = Front \ {}_F\|_B \ Back$$

where:

$$\begin{aligned} F &= \{forward, backward, nod, neigh\} \\ B &= \{forward, backward, wag, kick\} \end{aligned}$$

F and B are the alphabets of *Front* and *Back* respectively. If, in fact, the

processes *Front* and *Back* were defined as:

$$\begin{aligned} \mathit{Front} &= \mathit{forward} \rightarrow \mathit{neigh} \rightarrow \mathit{Front} \\ &\quad \Box\ \mathit{nod} \rightarrow \mathit{Front} \\ \mathit{Back} &= \mathit{backward} \rightarrow \mathit{kick} \rightarrow \mathit{Back} \\ &\quad \Box\ \mathit{wag} \rightarrow \mathit{Back} \end{aligned}$$

then the process *Horse* will never perform the events *forward* or *backward*, but it will simply *nod* and *wag* forever!

1. Why can the process *Horse* never perform the events *forward* and *backward* when *Front* and *Back* are defined as above? Can *Front* ever perform a *neigh* event? Can *Back* ever perform a *kick* event?

2. Implement the processes *Horse*, *Front* and *Back* in $\mathrm{CSP_M}$ as they are defined above, including the alphabet sets. Load your file into FDR and use Probe to verify whether your script is free of errors.

   Hint: Don't forget to define the events. For example, the `forward` and `backward` events can be declared by `channel forward, backward` (channels with no parameters are simply events). Note that ${}_F\|_B$ can be written in $\mathrm{CSP_M}$[1] as `[F || B]`.

3. In the lectures, you have been introduced to the process $RUN(X)$. Define the process $RUN(\{nod, wag\})$ in $\mathrm{CSP_M}$.

   Your process *Horse* should refine $RUN(\{nod, wag\})$ (why?). Check that this is indeed the case with FDR.

   Hint: There is no direct equivalent to $?ev : X \rightarrow \ldots$ in $\mathrm{CSP_M}$, but it can be simulated by using the *replicated external choice* operator:

   ```
   P = [] ev : X @ ev -> ...
   ```

   This evaluates `ev -> ...` for each value of `ev` in `X`, and then combines the resulting processes together using `[]`. For example, given the definitions from Part 1, `input?x:{1,4} -> STOP` is equivalent to `[] x : {1,4} @ input.x -> STOP`. Use the above operator to define a version of `RUN` that takes a set of events using the above operator.

4. Define a new pair of processes $Front_1$ and $Back_1$ with the same alphabets, F and B, respectively (as defined above), so that the overall behaviour of $Horse_1$ (defined analogously to *Horse*) is the same as:

   $$PH_1 = \mathit{neigh} \rightarrow \mathit{forward} \rightarrow \mathit{kick} \rightarrow \mathit{backward} \rightarrow PH_1$$

   Implement the process PH_1 in $\mathrm{CSP_M}$ and use FDR to check that your new process $Horse_1$ is equivalent to PH_1. (Note, A is (traces) equivalent to B iff $A \sqsubseteq_T B$ and $B \sqsubseteq_T A$.)

[1] The FDR manual has a complete list of $\mathrm{CSP_M}$ operators and their 'blackboard' equivalents at `http://www.cs.ox.ac.uk/projects/fdr/manual/cspm/processes.html`

5. Define a new pair of processes $Front_2$ and $Back_2$ with the same alphabets, F and B, respectively (as defined above), so that the overall behaviour of $Horse_2$ (defined analogously to $Horse$) is the same as:

$$\begin{aligned} PH_2 = \; & forward \rightarrow neigh \rightarrow PH_2 \\ & \Box \; backward \rightarrow kick \rightarrow PH_2 \end{aligned}$$

   Implement the process PH_2 in $\mathrm{CSP_M}$ and use FDR to check that your new process $Horse_2$ is equivalent to PH_2.

6. Define a new pair of processes $Front_3$ and $Back_3$ with the same alphabets F and B respectively (as defined above), so that the overall behaviour of $Horse_3$ (defined analogously to $Horse$) is the same as:

$$\begin{aligned} PH_3 = \; & neigh \rightarrow wag \rightarrow forward \rightarrow PH_3 \\ & \Box \; wag \rightarrow neigh \rightarrow forward \rightarrow PH_3 \end{aligned}$$

   Implement the process PH_3 in $\mathrm{CSP_M}$ and use FDR to check that your new process $Horse_3$ is equivalent to PH_3.