CS375 Practical Class 5 — Specifications

This practical class uses FDR to investigate some more specifications.

Getting Started

1. Connect to `tartan` as usual, move to your CS375 directory, and start an editor.
2. Type `fdr2 &`. You should see two windows: one entitled "FDR 2.11", and a file chooser window.

The Level Crossing

1. Copy the file `/CS/ftp/pub/CS375/crossing1.fdr2` to your directory, and load it into your editor and FDR. It contains the definitions for the first version of the level crossing from lectures. There are four assertions, corresponding to safety of *SYSTEM* and *SAFE_SYSTEM*, expressed in two different ways. Check all the assertions.
2. Notice that when all the events except *crash* are hidden, and *SYSTEM* fails to satisfy the specification, selecting "Debug" from the "Assert" menu does not show all of the trace; all the events except *crash* are replaced by "_tau", which is ASCII for τ. This is what happens when events are hidden; they get replaced by τ, which is a "silent" or "internal" event. Therefore there seems to be less information available about the trace which causes the specification to fail. However, by double-clicking on the process definitions which are shown on the left of the debug window, it is possible to see the actual events, generated by parts of the system, which correspond to the τ events. Try this, and see that in fact you can find out just as much about the behaviour of the process as you could when the events were not hidden.
3. The *CONTROL* process is defined in a particular way, but there are other possibilities. The definition in the file (and in the lecture notes) raises the gate immediately after a train has gone through the crossing. However, at a real level crossing, the gate stays down if another train is approaching. Change the definition of *CONTROL* so that it works in this more sophisticated way, and check the assertions again.

The Other Level Crossing

1. Copy the file `/CS/ftp/pub/CS375/crossing2.fdr2` to your directory, and load it into your editor and FDR. It contains the definitions for the second version of the level crossing, which does not use the *crash* event. Check the assertions again.
2. As in the previous section, change the definition of *CONTROL* so that the gate stays down if another train is approaching, and check the assertions again.

Peterson's Algorithm

1. Copy the file `/CS/ftp/pub/CS375/peterson1.fdr2` to your directory, and load it into your editor and FDR. It contains the (lengthy!) CSP definition of Peterson's algorithm, as described in lectures. Define a process *SPEC* so that $SPEC \sqsubseteq_t SYSTEM$ is a specification of mutual exclusion between *P1* and *P2*, and use FDR to check that Peterson's algorithm does guarantee mutual exclusion.

2. Modify the definitions in `peterson1.fdr2` so that they model one of the mutual exclusion algorithms from Assessed Coursework 1, and use FDR to test whether mutual exclusion is guaranteed. Try all three algorithms. (The Pascal FC code for the three algorithms can be found in `mutex1.pfc`, `mutex2.pfc` and `mutex3.pfc`).

3. The file `/CS/ftp/pub/CS375/peterson2.fdr2` contains a shorter CSP definition of Peterson's algorithm. The use of channels instead of individual events makes the definitions of the processes and alphabets smaller. (It would be possible to shorten the definitions still further, but I haven't bothered for now; notice that *FLAG1* and *FLAG2* are essentially the same.) Read the definitions and see how they correspond to version 1.

Dekker's Algorithm

1. The file `/CS/ftp/pub/CS375/dekker.pfc` contains a Pascal FC implementation of Dekker's Algorithm, the first mutual exclusion algorithm to be discovered. Produce a CSP model of Dekker's Algorithm, following the example of either `peterson1.fdr2` or `peterson2.fdr2`, as you prefer. Check that mutual exclusion is guaranteed.