Heriot-Watt University

School of Mathematical and Computer Sciences

Distributed Systems Programming F29NM1

SPIN Exercise Sheet 1

Andrew Ireland

The aim of these exercises is to introduce you to both the Promela modelling language and the SPIN Design Verification tool. While Promela and SPIN were developed in order to support the verification of protocols, they have been used extensively across a wide range of software and hardware design verification problems. To get started make your own copy the Promela programs that exist in `~air/Spin/Examples/`.

Exercise 1

Consider the "Hello World" program:

```
proctype hello(){ printf("Hello\n") }
proctype world(){ printf("World\n") }
init { atomic { run hello(); run world () } }
```

Note that the use of `atomic` within the `init` process ensures that instances of `A` and `B` are created at the same time. Use this program to try out XSPIN's simulation capabilities. That is, within your home directory type `xspin` – this creates a new window called `SPIN CONTROL`. Now follow the steps:

Step 1: Using the `File` menu select the `Open` option – this creates a new window called `Open`.

Step 2: From the `Open` window open the `hello` file.

Step 3: Now using the `Run` menu select the `Set Simulation Parameters` option – this creates a new window called `Simulation Options`.

Step 4: Note that under `Simulation Style` component, the seed used in generating a simulation can be modified. For now simply select the `Start` option (bottom right hand corner) – this creates three new windows, `Simulation Output`, `Data Values` and `Message Sequence Chart`.

Step 5: From the `Simulation Output` window select the `Run` option – this starts a simulation based upon the `hello` model. Note that within the `Simulation Output` window the effect of the print statements will be displayed. Note that the `Data Values` window provides information on the value of global variables and channels – this will be useful in the next exercise. The `Message Sequence Chart` contains the trace of the inter-process communications – again this will be useful in later exercises.

Try re-running the simulation using a different seed value (step 4). On each simulation run check to see the order in which the strings "Hello" and "World" are displayed within the `Simulation Output` window. What effect do the different seed values have?

Exercise 2

Consider the following Promela program:

```
byte x = 2, y = 3;
proctype A(){x = x + 1}
proctype B(){x = x - 1; y = y + x}
init { atomic{run A(); run B()} }
```

Again run this program using different seed values. What effect do the different seed values have on the output displayed within the `Data Values` window?

Exercise 3

Consider the following Promela program:

```
byte state = 1;
proctype A() { (state == 1) -> state = state + 1 }
proctype B() { (state == 1) -> state = state - 1 }
init { atomic{run A(); run B()} }
```

Again run this program using different seed values. What effect do the different seed values have on the output displayed within the `Data Values` window? Repeat the same experiment using the following modified Promela program:

```
byte state = 1;
proctype A()
{ atomic{ (state == 1) -> state = state + 1 } }
proctype B()
{ atomic{ (state == 1) -> state = state - 1 } }
init { run A(); run B() }
```

Again what effect do the different seed values have on the output displayed within the `Data Values` window?

Exercise 4

Extend the "Hello World" program with a process type called `control`. Define `control` so that it ensures that the `hello` process always performs its print statement before the print statement associated with the `world` process. You should use two channels to achieve the desired behaviour. Be sure that you run the syntax checker before you attempt to invoke the simulator.

Exercise 5

Consider again the integer division (`division`) and factorial (`fact`) programs given in lecture. Can you account for the MSCs generated by each program? Can you account algorithmically for any differences between the general structure of the MSC's generated by the programs?

Exercise 6

Consider a vending machine that contains chocolate bars, both milk and plain. To select a milk chocolate bar one inserts 20 pence while in the case of plain chocolate the cost is 50 pence. Define a process type called `vender` that models the vending machine. Moreover, define a process called `customer` that models a "chocoholic", *i.e.* someone who continuously consume chocolate bars. Assume that the `vender` has a limitless supply of chocolate while the `customer` as a limitless appetite for chocolate, both plain and milk. Again you should use message channels to model the various lines of communication between the `customer` and `vender`, *i.e.* coins and chocolate.

Exercise 7

Refine your model of the vending machine by limiting the number of chocolate bars that are initially held, *i.e.* 10 milk and 5 plain chocolate bars. In addition, include a coin box that models the amount of money held within the vending machine. Assume that the coin box is initially empty and that once all the chocolate bars are sold that no money is accepted.

Exercise 8

Add a local assertion to your vending machine model that expresses the fact that there exists an invariant relationship between the amount of money and chocolate it holds. Use the simulator to check your invariant.

Exercise 9

Make one final refinement to your vending machine model so that the `customer` only has 450 pence to spend. Furthermore define a system invariant which states that the amount of money in the overall system is always 450 pence. Check your solution using the simulator.

Exercise 10

Consider a simple water storage system that involves, `sensors`, a `user` and `inlet` and `outlet` devices. The `sensors` measure the water level within a storage device. The `outlet` device provides water for the `user`. The demand for water by the `user` is not constant, *i.e.* at each moment in time the `user` decides randomly whether or not to request water. Whenever the water level reaches 20 units the `sensors` close the `outlet` and open the `inlet`. This causes the water level to rise. Once the water level reaches 30 units the `inlet` is closed and the `outlet` is opened again. Model the water storage system using distinct processes to model the `sensors`, `user`, `inlet` and `outlet`. Include an assertion to ensure that the water level is always within the range 20 to 30 units. Evaluate your model using the simulator.