CSc72010: Assignment 1

Nancy Griffeth

February 2, 2006

1 Using the IOA toolkit.

With the IOA toolkit, you can write algorithms in an executable language and simulate their execution. We summarize the steps you take to write a new algorithm below. The initial assignment requires only that you modify an IOA definition that I have provided with the assignment (`Leader.ioa`). Thus, rather than defining your own automaton, you will modify the given automaton.

1. First define your automaton in IOA language and save it in a `.ioa` file (`<filename.ioa>`). To verify that your syntax is correct, simply run `ioaCheck <filename.ioa>`

2. Once the syntax is correct, you can create an intermediate file with the suffix `.il`. The simulator can use this file. To create the intermediate language file, run the command:

   ```
   ioaCheck -il <filename.ioa> > <destfile>.il
   ```

   The output is of ioaCheck is written to stdout. The above command redirects the output to a file of your choosing, with the extension `.il`. This will overwrite any pre-existing file named `<destfile>.il` in the current directory.

3. Finally, you run the simulator. You can run the simulator from either the ioa file (the simulator will take longer to run, because it has to do the syntax check first) or the intermediate language file.

   To run the simulator directly from the ioa file, use the command:

   ```
   sim <number of steps> <nameOfAutomaton> <filename>.ioa
   ```

   To use the intermediate file, the command is:

   ```
   sim <number of steps> <nameOfAutomaton> <ILfilename>.il
   ```

   You can redirect the output to a file in the usual manner, i.e.

```
sim <number of steps> <nameOfAutomaton> <ILfilename>.il > <outputFilename>
```

4. You can limit the output to transitions by using the -outputTrans option, or further limit it to showing only external transitions by the -outputTraces option. That is,

   ```
   sim -outputTrans <number of steps> <automaton> <file>.il
   ```

   or

   ```
   sim -outputTraces <number of steps> <automaton> <file>.il
   ```

   Of course, either of the above can also be redirected to a file.

   If you need more sophisticated filtering, the raw output includes all the details that are available, including the state changes. If you need to look at information about particular state changes, you can do pattern-matching to select lines of the file using `grep` for simple pattern matches or `egrep` for general regular expressions. If you have saved the output of a simulation to `file`, the following command will show all transitions:

   ```
   egrep input\|output\|internal <file>
   ```

   You can also pipe the output directly into the following command:

   ```
   egrep input\|output\|internal  -
   ```

   The above commands should write all lines containing either the string "input" or the string "output" or the string "internal" (these are the three kinds of transitions).

   If you are interested instead in a particular state variable, e.g. `buffer`, you could use the command:

   ```
   grep buffer <file>
   ```

For more command options, type `sim` or `ioaCheck` without any parameters.

2 The Algorithms

`Assignment1.zip` contains some examples of IOA algorithms, complete with schedules for simulation.

2.1 Random number generator.

This example picks a random number from 1 to 100 and "outputs" it. The schedule has the simulator running forever. The internal action "action1" chooses an integer; the output action "action2(n)" is fired each time an integer has been chosen.

2.1.1 Instructions.

The file defining the automaton and the schedule:

```
Chooser.ioa
```

The command to run the simulation:

```
sim <n> Chooser Chooser.ioa
     where <n> is the number of steps it will run.
```

The following command will show all transitions for the first 100 steps:

```
sim -outputTrans 100 Chooser Chooser.ioa
```

If you prefer to see only the outputs (the numbers chosen) use:

```
sim -outputTraces 100 Chooser Chooser.ioa
```

2.2 Leader Algorithm

The automata in this example work together to pick a single automaton, which is designated the leader (and fires the output action "leader"). The scheduler assigns random ID's to the automata, and the one with the larget ID will become the leader. The scheduler will run forever, but after the leader has been chosen takes no steps. You will usually have to stop it with `ctrl-C`.

2.2.1 Instructions.

The file defining the automata and the schedule:

```
Leader.ioa
```

Note that the file contains both Process and Channel automata; they are composed into a Leader automaton.
The command to run the simulation is:

```
sim <n> Leader Leader.ioa
     where <n> is the number of steps it will run.
```

The following command will show all states and actions for the first 100 steps:

```
sim 100 Leader Leader.ioa
```

This is plenty to run the entire simulation. You will have to terminate with `ctrl-C`.

3 Problems

1. What is the minimum number of transitions you must see to see the entire Leader simulation, with 5 processes and 5 channels? What is the maximum number of transitions? Suppose you have 10 processes and 10 channels? Generalize to `n` processes and `n` channels. This is a measure of the time required to run the leader election algorithm, i.e., its *time complexity*.

2. Re-write the Channel automaton so that it re-orders messages. Hint: You can define the buffer as a Set[Int] instead of a Seq[Int]. The relevant operations on a set are:
   `delete(X, S)` - delete the element X from S
   `insert(X, S)` - insert the element X from S
   `X in S` - return `true` if X is in S
   Does this affect the minimum and maximum number of transitions it takes to elect a leader? Why or why not?

3. (Extra credit.) Suppose you use a Channel that loses messages. What is the minimum and maximum number of transitions in this case? Suppose the Channel duplicates messages. What is the minimum and maximum number of transitions?