CS405 Introduction to Empirical Modelling 2009

Labsheet 1: Modelling with definitive scripts preliminaries

Using definitive notations to represent state

The notion of a definitive script

A *definitive script* is a set of definitions (note carefully that 'set' is actually potentially less misleading than 'script', which suggests an activity rather than a description, but the term "definitive script" has been used extensively in EM publications). The LHS of a definition is an *observable*, and the definition expresses a *dependency* amongst observables. Informally, when we redefine the value of one observable the values of any observable defined in terms of this observable are 'instantly' updated. (What 'instantly' means here is a subtle issue to be explored in the module.) The observables are of different types, each associated with a *definitive notation* used to formulate the formulae that define values. Different definitive notations have different kinds of values and operators to express one value in terms of others.

Preliminaries

First create a personal directory `cs405practical` for your CS405 practical work. Copy the contents of the directory

```
/dcs/emp/empublic/teaching/cs405/lab1
```

into your `cs405practical` directory. On Linux, you can do this by making `cs405practical` your current directory and using

```
cp -r /dcs/emp/empublic/teaching/cs405/lab1 .
```

Exercise 1: Using the EDEN interpreter to create a definitive script

Our first concern is to ensure that everyone is familiar with basic use of the EDEN interpreter. We shall introduce EDEN by demonstrating the following features:

1. The interpreter is launched by
   - typing `tkeden` (on Linux)
     OR
   - executing `tkeden.exe` (on Microsoft Windows)

   This sets up an *input window*.

2. At any given time, the current state of the interpreter is specified by a script. Initially, you can think of this script as empty: it has no definitions in it.

3. When modelling state in the most primitive way, you enter the definitions by hand via the input window. At any stage, you can enter new definitions or change existing definitions, and press `Accept`

4. Each definition has a variable (or "observable") on its LHS, and a formula as its RHS. It specifies a dependency.

5. Definitions in `eden` are specified using the keyword: **is**. If you want to define the value of an observable to be the current value of a formula (as in traditional programming assignment), you replace **is** by **=**.

6. You can inspect the current set of values of observables and dependencies using `writeln` and **?**

Exercise 2: Loading and experimenting with an existing script

Most EDEN scripts contain definitions for observables with visual significance. We shall illustrate this by playing with a simple script that uses the line drawing notation `donald`. To enter a `donald` definition, you either precede the definition by `%donald` or select the `%donald` radio button on the EDEN input window. There are some standard potential sources of confusion: unlike an `eden` definition, a `donald` definition is *not terminated by a semi-colon* (;) and uses the symbol "=" to denote a *definition* rather than an assignment.

Launch the EDEN interpreter. Use the **Open** option on the **File** menu in the input window to navigate to your `cs405practical` directory and double-click on the name of the file `room.d` to place the set of definitions in the file `room.d` in the EDEN input window. Press `Accept` to load the definitive script in `room.d`.

1. Open the file `sampledefns` in a text editor, and test out each of the definitions listed there in turn by copying it into the input window and pressing `Accept` or the keyboard shortcut `Alt-A`. Consider how each definition helps you to assess the plausibility of the computer artefact as a model of a room.

2. What is unsatisfactory about the redefinition that appears to restore the table to normality? How would you make this obvious? How would you restore the table in a more satisfactory way?

3. Look at the definitions of the desk on pages 2 and 3 of the `room.d` script. Type redefinitions directly into the tkeden input window to change the size and position of the desk. Note the dependency of the drawer size on the desk. Can you make a better model of the opening and shutting of the desk drawer? (To see a sample answer, inspect the file `fixdesk.e`.)

4. Change the width of the door. Make it a sliding door. Can you make the opening of the door (in either version) more realistic than having only two states?

5. Using `donald` definitions of the form

   ```
   point p, q, r
   boolean b
   p = if b then q else r
   ```

   arrange for the lamp to appear to be on when the door is open, and off when it is closed.

Questions

1. Examine sample observables drawn from the room script. Which observables are subject to change, and what sort of agent would be able to change them?

2. What role do dependencies play in determining what we think an EM artefact might refer to?

3. Which of the following would you say best expresses the sense in which the EM artefact represents a room:

   - the set of current definitions.
   - those sets of definitions that can be derived from the current definitions by performing a standard repertoire of meaningful redefinitions (such as `door/open=true`).
   - those sets of definitions that can be derived from the current definitions by open-ended experimental interactions that admit an imaginative interpretation as actual interactions with a room.
