MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring Semester, 1998

Problem Set 11

- Issued: Tuesday, 21 April 1998 (and copies distributed in recitation on Wednesday, 23 April 1998)
- Tutorial preparation for: Week of 27 April 1988
- Written solutions due: in recitation Friday, 1 May 1998
- Reading: Read chapter 5 through section 5.3 (pp. 491-546) before lecture on Tuesday, 28 April 1998. Read section 5.4 (pp. 547-566) before lecture on Thursday, 30 April 1998.
- Code: The following code (attached) should be studied as part of this problem set:
  - `load-amb.scm`—loader for AMB evaluator system
  - `syntax.scm`—Syntax support for evaluator experiments
  - `evdata.scm`—Evaluator data structures
  - `rep-support.scm`—support for driver loop
  - `let.scm`—support for LET (as noted in footnote 56, p.428)
  - `ambeval.scm`—actual code for AMB evaluator
  - `puzz.scm`—an example puzzle coded for the AMB evaluator and other useful goodies you may need

Nondeterministic Computing

This problem set is entirely based on the nondeterministic Scheme interpreter described in section 4.3 of SICP. All of the exercises are from the book. The programming problems will require use and some modification of the evaluator we supply as part of the problem set.

Tutorial Exercise 1: Do Exercise 4.35 (p.417)

Tutorial Exercise 2: Do Exercise 4.36 (p.417)

Tutorial Exercise 3: Do Exercise 4.37 (p.418)

Programming with AMB

In this problem set we will be using a search evaluator written in Scheme. The file `ambeval.scm` contains an interpreter for AMBScheme—Scheme augmented with `amb`. The interpreter is a procedure-tree interpreter, that separates analysis from execution. It has the same basic structure as the one in SICP section 4.1.7, but there are major differences. The AMBScheme evaluator is explained in section 4.3.

The user interface (ha!) to the AMBScheme system is pretty crude. To initialize the evaluator use the incantation `(define the-global-environment (setup-environment))`. If you type `(driver-loop)`, you will find yourself typing at the driver loop. Please note that AMBScheme running in Scheme has no error system of its own. If you hit an error or if you interrupt your program, you will bounce back into Scheme.

The driver loop uses the prompt `;;; Amb-Eval input:` instead of the ordinary Scheme prompt.

Start up the interpreter and try a few simple expressions. If you bounce out into Scheme, you should quit out of the error by `(RESTART 1)` and re-enter the interpreter by typing `(driver-loop)`. If you get hopelessly fouled up, you can reinitialize your global environment, but you will lose any definitions you have made.

Loading problem set 9 will get a working evaluator loaded. If you start the AMBScheme evaluator and then load up the example program, `puzz.scm`, using `ambload`, you should obtain the following behavior:

```
(ambload "puzz.scm")
;Value: done

(driver-loop)

;;; Amb-Eval input:
(multiple-dwelling)
;;; Starting a new problem
;;; Amb-Eval value:
((baker 3) (cooper 2) (fletcher 4) (miller 5) (smith 1))

;;; Amb-Eval input:
try-again
;;; There are no more values of
(multiple-dwelling)

;;; Amb-Eval input:
```

When doing the programming problems for code to be executed by the AMB evaluator, a good strategy is to edit the file `puzz.scm`, adding your new code. You can then reload the file and restart the evaluator to do your tests.

Computer Exercise 1. (Exercise 4.38 on p.419) Run the program to get the result shown above. Then, prepare a modified version that relaxes the requirement that Smith does not live on a floor adjacent to Fletcher's. Run your modified program. Is the solution still unique? If not, how many solutions are now possible?

Computer Exercise 2a. (Exercise 4.39 on p.419) Does the order of the restrictions affect the answer? the time to find an answer? If you think it matters, demonstrate a faster program obtained from the given one by reordering the restrictions. If you think it does not matter, argue your case.

Computer Exercise 2b. (Exercise 4.40 on p.419) Write and demonstrate a very efficient search procedure for this problem based upon only generating those possibilities that are not already ruled out by previous filtrations. Hint: This will require a nest of `let` expressions.

Computer Exercise 3. Do Exercise 4.43 (p.420)

Understanding the Interpreter.

Computer Exercise 4. Do Exercise 4.50 (p.436)

Computer Exercise 5. Do Exercise 4.51 (p.436)

Computer Exercise 6. Do Exercise 4.52 (p.436)

Computer Exercise 7. Do Exercise 4.53 (p.437)