CS655 — Homework Assignment 1 (solutions due Tue Jan 31)

Wes Weimer

January 24, 2006

Exercise 1: Bookkeeping. Send me email with your name and indicate whether you are taking the class for a letter grade, taking it pass/fail, or just sitting in. In addition, indicate one thing you like about the class and one thing you would change about it. I will use the email address you provide to send bulletins related to the course (e.g., notifications of updates or corrections to future homeworks). You can simplify things by attaching a PDF of your written answers to this assignment and your source code for question 5 (see below) in that same email.

Exercise 2: Language Design. Comment on some aspect from Hoare's *Hints On Programming Language Design* that relates to your programming experience. Provide additional evidence in favor of one his points and against one of his points. Do not exceed three paragraphs. Both your ideas and also the clarity with which they are expressed (i.e., your English prose) matter.

Exercise 3: Set Theory. This excercise is meant to help you refresh your knowledge of set theory and functions. Let X and Y be sets. Let $\mathcal{P}(X)$ denote the powerset of X (the set of all subsets of X). There is a 1-1 correspondence (i.e., a bijection) bewteen the sets A and B, where $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$. Note that A is a set of functions and B is a (or can be viewed as a) set of relations. This correspondence will allow us to use functional notation for certain sets in class. This is Exercise 1.4 from page 8 of Winskel's book. *Do one of the following:*

- Demonstrate the correspondence between A and B by presenting an appropriate function and proving that it is a bijection. For example, you might construct a function $f : B \to A$ and prove that f is an injection and a surjection.

- Write "I understand this background material" but do not do the problem. You will receive full credit for this question. We use the Honor system.

Exercise 4: Simple Operational Semantics. Consider the IMP language discussed in class, with the Aexp sub-language extended with a division operator. Explain what changes must be made to the operational semantics (big-step only). Write out formally any new rules of inference you introduce.

Exercise 5: Language Feature Design. Consider the IMP language with a new command construct "`let` $x = e$ `in` c". The informal semantics of this construct is that the Aexp e is evaluated and then a new local variable x is created with lexical scope c and initialized with the result of evaluating e. Then the command c is evaluated. We also extend IMP with a new command "`print` e" which evaluates the Aexp e and "displays the result" in some un-modeled manner but is otherwise similar to `skip`.

We expect (the curly braces are syntactic sugar):

```
x := 1 ;
y := 2 ;
{ let x = 3 in
  print x ;
  print y ;
  x := 4 ;
  y := 5
} ;
print x ;
print y
```

to display "3 2 1 5".

- Extend the natural-style operational semantics judgment $< c, \sigma > \Downarrow \sigma'$ with one new rule for dealing with the `let` command. Pay careful attention to the scope of the newly declared variable and to changes to other variables.
- Extend the set of redexes and contexts for the contextual-style operational semantics that we discussed in class to account for the `let` command.
- Download the Homework 1 code pack from the course web page. Modify `hw1.ml` so that it implements a complete interpreter for IMP (including `let` and `print`). Base your interpreter on IMP's large-step

operational semantics. The `Makefile` includes a "`make test`" target that you should use (at least) to test your work.

- Modify the file `example-imp-command` so that it contains a "tricky" IMP command that can be parsed by our IMP test harness (e.g., "`imp < example-imp-command`" should not yield a parse error).

- Rename `hw1.ml` to `your_last_name-hw1.ml` and rename `example-imp-command` to `your_last_name-imp-command.ml` and email them to me. Do not modify any other files. Your submission's grade will be based on how many of the submitted `example-imp-commands` it interprets correctly (in a manner just like the "`make test`" trials). If your submitted `example-imp-command` breaks the greatest number of interpreters (and more than 0!), you will receive extra credit. If there is a tie all tiers will receive the extra credit.