Lab: Calling continuations
CSC 312P, “Programming language implementation: the good parts”
Department of Computer Science · Grinnell College
November 2, 2015

Exercises

A. Create a subdirectory called CPS-EFFECTS in the directory you’re using for labs in this course and copy my implementation of the CPS-EFFECTS programming language into it. The directory /home/stone/courses/languages/code/CPS-EFFECTS contains the eight language-specific modules that you need.

- expvals-and-environments.scm
- stores.scm
- tokens.scm
- syntax-trees.scm
- scanner.scm
- parser.scm
- translator.scm
- interpreter.scm

If you don’t already have the four utility modules test.scm, natural-numbers.scm, list-of.scm, and character-sources.scm, you’ll also need to copy them from the directory /home/stone/courses/languages/code into the parent directory of CPS-EFFECTS.

B. Instead of letcc-expressions and throw-expressions, Scheme offers a built-in procedure called call-with-current-continuation (or, for short, call/cc—both names are predefined in Scheme). It is a unary procedure that takes another unary procedure as argument. When invoked, call/cc in turn invokes the procedure that it is given as its argument, providing the continuation of the call that invoked it as the argument to the procedure. Thus the effect is to bind the continuation for the invocation of call/cc to the parameter of the procedure it is given and then to evaluate the body of that procedure. If the procedure is not interrupted and yields a value, its value becomes the value of the call to call/cc.

Here’s a typical example:

```scheme
(define product-of-reciprocals
  (lambda numbers
    (call-with-current-continuation
     (lambda (k)
       (let kernel ((rest numbers))
         (cond ((null? rest) 1)
               ((zero? (car rest)) (k 0))
               (else (/ (kernel (cdr rest)) (car rest))))))))
```

This procedure takes any number of arguments, each of which must be a number, and finds the product of their reciprocals, or 0 if any of the elements of the list is 0. It avoids dividing by zero by testing each element first. If it finds a zero, it directly invokes the continuation for the original call, discarding both the computation done so far and any pending divisions.

Formulate a rule, similar to the rules for letcc- and throw-expressions on page 232 of the text, for callcc-expressions, assuming that callcc (proc (k) ...) works in CPS-EFFECTS as (call/cc (lambda (k) ...)) works in Scheme.
C. Add `callcc`-expressions to the interpreter for CPS-EFFECTS.