

Class 17: Continuation Basics

Held: Wednesday, February 28, 2007

Summary: Today we begin to explore continuations, one of the more interesting control mechanisms in Scheme.

Related Pages:

- EBoard.
- Reading: Paul Graham - Scheme Continuations and R. Kent Dybvig - Continuations and Continuation Passing Style.

Notes:

- I'll reserve some time at the start of class to discuss the upcoming homework.
- EC: G-Tones Concert on Friday at 7:30 in the Bucksbaum rotunda.
- EC: Henry Walker speaks on Google®'s PageRank® algorithm Thursday at noon.

Overview:

- Basics of Continuations.
- Some Examples.
- Some Applications of Continuations.
- Continuation-Passing Style.

Continuations: Some Basic Concepts

- Continuations are functions that represent “the state of the program” or “what happens next”
- Consider the following:
 $(+ 4 (* 3 (- (+ x x) a)))$
 - The continuation for $(* \dots)$ is “add 4” or $(\lambda (x) (+ x 4))$
 - The continuation for $(- \dots)$ is “multiply by 3 and then add 4”, or $(\lambda (x) (+ 4 (* 3 x)))$
 - The continuation for $(+ x x)$ is ...
- Every language has an implicit notion of continuations
 - After all, until the program finishes, at any point you have a notion of “the next code to execute”
- Scheme makes these continuations *first-class objects*:
 - You can get them.
 - You can send them as parameters to other functions.
 - You can call them.
 - Etc.
- When you call a continuation, you replace the current continuation with the called continuation

- That is, instead of doing what appears to come next, you do what the continuation says comes next.
- To get a continuation, use `(call-with-current-continuation (lambda (cont) ...))`
 - This wraps up the current continuation and pass it to the lambda.
 - `(call/cc (lambda (cont) ...))` is an alias for `call-with-current-continuation`.
- A nice alternative is `(let/cc k body)`
 - This wraps up the current continuation, calls it k, then executes the body.
 - Same as `(call/cc (lambda (k) body))`

Two Examples

- Dybvig's product example, which we'll build piece by piece.
 - The standard version (regular recursion, no special case for 0)
 - Add a special case for 0
 - Add continuations
- Escaping from a deep computation

```
(+ 3 (* 4 (- (/2 x) 5)))
```

 - With no error checking
 - With error checking at every step and a special return value
 - With continuations.

Some Applications

- Experienced Scheme programmers use continuations for a variety of applications.
- Continuations provide one mechanism for indicating *Exceptions*.
 - You pass in the top-level continuation.
 - When you want to exit early, you call that top-level continuation.
- Continuations can provide a mechanism for pausing and resuming code.
 - First, create two global continuations
 - `exit`, which lets you return to the top level
 - `resume`, which lets you restart the code at the stopped point.
 - When you want to pause, update `resume` and then call `exit`.
 - Graham uses this as a way to pause and restart tree traversal.
- Continuations provide a mechanism for coroutines (that is, having multiple routines share computation)
 - Similar technique to pause/resume, except that `exit` resumes the other routine.
- Continuations provide a nice mechanism for saving state on the Web; we'll discuss it in the next class.

Continuation-Passing Style

- If a functional language does not provide explicit continuations, you can use a programming style that makes the continuation an explicit parameter to every function (typically, the last parameter).
- This style is called *Continuation-Passing Style* or *CPS*.
- For example, the CPS version of the two-parameter plus might look like

```
(define cps-plus
  (lambda (x y cont)
    (cont (+ x y))))
```

- Consider the example from above. (+ 4 (* 3 (- (+ x x) a)))
- Using CPS, we might write

```
(cps-plus x x (lambda (tmp1)
  (cps-minus tmp1 a (lambda (tmp2)
    (cps-times 3 tmp2 (lambda (tmp3)
      (cps-plus 4 tmp3 (lambda (tmp4)
        (display-result tmp4))))))))))
```

- Notice how close this looks to assembly code.
- Many Scheme compilers (and some interpreters) convert to continuation-passing style.