

## Class 18: Continuations and Web Servers

**Held:** Friday, March 2, 2007

**Summary:** Today we consider the application of continuations to Web services.

### Related Pages:

- EBoard.
- Reading: Christian Queinnec - The Influence of Browsers on Evaluators or, Continuations to Program Web Servers and Samuel Rebelsky - webcont.scm.

### Due

- HW5.

### Notes:

- Because I will be distributing the exam on Monday, there is no homework until after break.
- Instead of doing a reading for Monday, answer the following questions: (a) What is polymorphism? (b) What is overloading? (c) How do the two relate?

### Overview:

- Continuations and the Web.
- Sam's sample code.
- Applications of continuations, revisited.
- Continuation-Passing Style.

## Continuations and Web Servers

- While continuations provide an interesting control mechanism, they did not find widespread usage.
  - They are not really implemented in non-Lisp languages.
  - They are complicated to understand (as you've seen from the prior readings)
- The advent of the Web has led some folks (Queinnec may be the first) to suggest that continuations provide an appropriate mechanism for handling the "statelessness" of the Web.
- The strategy: Since each page in a sequence of pages represents a state of a computation, we can use continuations to better capture that state.
  - Programmers traditionally find some way to represent that state manually.
  - Continuations automate the process.
- By making the continuation part of the URL, and saving the continuation on the Web server, we can even let people send partial sessions to their colleagues.

## My Sample Code

- We can think of a typical session with a Web server as being a series of prompts from the server and responses from the user.
  - The prompts can be fields or buttons.
  - Subsequent prompts can be based on prior answers.
- In my code, I model the interaction with the client with two functions, `prompt`, which prompts the user, and `result`, which displays a result.
- A Scheme design strategy: To permit testing,
  - I make multiple versions of `prompt` and `result` (one for shell-based interaction and one for a simulation of Web-based interaction).
  - I make `prompt` and `result` aliases to whichever one I currently want to use.
  - `(local)` and `(web)` let us switch between versions.
- The Web-based version of `prompt` would normally:
  - Capture the continuation that expects the result of the page.
  - Build the page that includes the prompt (and the subsequent).
  - Make the submit button on the page call the continuation.
- We simulate that by printing the continuation and the prompt.
- The next request on the Web would use the continuation and the value.
- We simulate that with the `resume` function.

## 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.