Assignment 8: Registerization and Trampolining

> A trampoline is strong enough to catch you before you hit the ground, but not so cushy that you can live on it forever
>
> —*Thomas Friedman*

Note

Do consult your class notes and any files distributed to you from lecture or lab.

As usual, you can use the a8-student-tests.rkt tests file to test your submission, although these tests are not exhaustive.

Assignment

Consider the following four procedures:

```
(define ack
  (lambda (m n k)
    (cond
      [(zero? m) (k (add1 n))]
      [(zero? n) (ack (sub1 m) 1 k)]
      [else (ack m (sub1 n) (lambda (v) (ack (sub1 m) v k)))])))

(define depth
  (lambda (ls k)
    (cond
      [(null? ls) (k 1)]
      [(pair? (car ls))
       (depth (car ls)
       (lambda (l)
  (depth (cdr ls)
         (lambda (r)
    (let ((l (add1 l)))
      (k (if (< l r) r l)))))))]
      [else (depth (cdr ls) k)])))

(define fact
  (lambda (n k)
    ((lambda (fact k)
       (fact fact n k))
     (lambda (fact n k)
       (cond
         [(zero? n) (k 1)]
         [else (fact fact (sub1 n) (lambda (v) (k (* n v))))]))
     k)))

(define pascal
  (lambda (n k)
    (let ((pascal
           (lambda (pascal k)
             (k (lambda (m a k)
    (cond
      [(> m n) (k '())]
      [else (let ((a (+ a m)))
       (pascal pascal (lambda (f) (f (add1 m) a (lambda (v) (k (cons a v)))))))]))))))
      (pascal pascal (lambda (f) (f 1 0 k))))))
```

Here are examples of how to call these procedures:

```
> (ack 2 2 (empty-k))
7
> (depth '(1 (2 (3 (4)))) (empty-k))
4
> (fact 5 (empty-k))
120
> (pascal 10 (empty-k))
(1 3 6 10 15 21 28 36 45 55)
```

Part I

You should write a trampolinized version of the original programs and write the corresponding drivers (as mentioned in Tuesday's lecture).

```
> (ack-tramp-driver 2 3)
9
> (depth-tramp-driver '((((a) b (c (d))) e)))
5
> (fact-tramp-driver 3)
6
> (pascal-tramp-driver 10)
(1 3 6 10 15 21 28 36 45 55)
```

In the case of a trampolinized program, the job of the driver will be to grab an initial continuation and start the trampoline.

Part II

For each of the four initial programs, you should write a registerized version of the original **without** trampolining.

```
> (ack-reg-driver 2 2)
7
> (depth-reg-driver '(1 (2 (3 (4)))))
4
> (fact-reg-driver 5)
120
> (pascal-reg-driver 10)
(1 3 6 10 15 21 28 36 45 55)
```

In the case of a registerized program, the job of the driver will be to populate the registers with initial values and call the registerized program. You don't need to trampoline your registerized programs.

Both Parts

Each of these programs will take the arguments taken by the original program.

As you do the assignment, keep the following points in mind:

- For the purposes of this assignment, you should always registerize or trampolinize the original program. Don't write a registerized version of a trampolinized program. You don't need to trampoline your registerized programs.
- All definitions must be *representation-independent with respect to continuations*. You must use a *data-structural* representation of continuations.
- **Make sure to name your drivers exactly as above, or we will not be able to grade your submission!**

When you are finished, place all of your code in a file named `a8.scm` and submit it to Oncourse.

Brainteaser

Trampolines can be used to execute multiple trampolinized programs simultaneously. Using a trampoline, write a procedure `rampoline` that takes three thunks containing trampolinized procedure calls, and executes them in random order, returning the value of the first to complete. Note that the procedures may go into an infinite loop. To show that `rampoline` works correctly, CPS and trampolinize the following definition of `fib`:

```
(define fib
  (lambda (n)
    (cond
      [(= n 0) 1]
      [(= n 1) 1]
      [else (+ (fib (sub1 n)) (fib (sub1 (sub1 n))))])))
```

You could use the following as a driver to your trampolinized fib:

```
(define fib-dr
  (lambda (n1 n2 n3)
    (call/cc
      (lambda (jumpout)
 (rampoline
     (lambda ()
       (fib n1 (empty-k jumpout)))
     (lambda ()
       (fib n2 (empty-k jumpout)))
          (lambda ()
       (fib n3 (empty-k jumpout))))))))
```

Just Dessert

A trampoline can be used to interleave executions of two trampolinized programs, we've seen how executing two such programs in a trampoline can return the first answer that completes. But what if we want /both/ answers?

Devise a way to return both answers, wherein:

- We still interleave executions of both programs with a trampoline.

- We get back a list with both answers.
- The answers are listed in the order they complete.
- We don't redo any work after finding the first answer.

You will probably need to define a `bi-trampoline`, a `bi-tramp-driver`, and a trampolinized functions of your choosing to test it. Automated tests are set to run `bi-tramp-driver` with two numbers.

```
> (fib-tramp-driver 3 4)
(3 5)
> (fib-tramp-driver 4 3)
(3 5)
> (fib-tramp-driver 6 6)
(13 13)
```

assignment-8.txt · Last modified: 2014/10/09 19:55 by jhemann