SICP section 4.3.3

January 11, 2008 at 18:38 Tags SICP

I've implemented the `amb` evaluator fully before starting with section 4.3 – it can be downloaded here

Exercise 4.50

I will use the existing `amb`, passing it a shuffled list of choices:

```
(defun ramb? (exp) (tagged-list? exp 'ramb))

(defun analyze-ramb (exp)
  (analyze-amb
    (cons 'ramb
          (shuffle-list (amb-choices exp)))))
```

`shuffle-list` is this naive[1] procedure:

```
(defun shuffle-list (lst)
  (sort lst #'(lambda (x y) (zerop (random 2)))))
```

And finally, this has to be added to the `cond` in `analyze.`:

```
  ((ramb? exp)
    (analyze-ramb exp))
```

Exercise 4.51

Adding this to the `cond` in `analyze.`:

```
  ((permanent-set? exp)
    (analyze-permanent-set exp))
```

And this is the implementation:

```
(defun permanent-set? (exp) (tagged-list? exp 'permanent-set!))

(defun analyze-permanent-set (exp)
  (let ((var (assignment-variable exp))
        (vproc (analyze. (assignment-value exp))))
    (lambda (env succeed fail)
      (funcall vproc
        env
        (lambda (val fail2)
          (set-variable-value! var val env)
          (funcall succeed
            'ok
            (lambda ()
              (funcall fail2))))
        fail))))
```

Note that it's very similar to analyze-assignment, except that it doesn't roll back the old value in the fail continuation passed to `vproc`.

Exercise 4.52

```
(defun analyze-if-fail (exp)
  (let ((pproc (analyze. (if-predicate exp)))
        (cproc (analyze. (if-consequent exp))))
    (lambda (env succeed fail)
      (funcall pproc
        env
        (lambda (pred-value fail2)
          (if (true? pred-value)
            pred-value
            (funcall fail)))
        (lambda ()
          (funcall cproc env succeed fail))))))
```

With the usual additions to the evaluator:

```
(defun if-fail? (exp) (tagged-list? exp 'if-fail))
```

And into `analyze.`:

```
  ((if-fail? exp)
    (analyze-if-fail exp))
```

Exercise 4.53

It prints:

```
((8 35) (3 110) (3 20))
```

Although the `let` form always fails (it calls `(amb)` as its last statement), the pairs get added into `pairs`, because `permanent-set!` doesn't roll assignments back from failed paths.

Exercise 4.54

```
(defun analyze-s-require (exp)
  (let ((pproc (analyze. (s-require-predicate exp))))
    (lambda (env succeed fail)
      (funcall pproc
        env
        (lambda (pred-value fail2)
          (if (not (true? pred-value))
            (funcall fail)
            (funcall succeed 'ok fail2)))
        fail))))
```

[1] It's naive because it's inefficient and doesn't produce a perfect shuffle. Rather, the shuffle depends on the sorting algorithm. However, for our needs here, this shuffle is fine.

For comments, please send me an email.

Back to top