6.001 SICP Spring 2004 5/7/2004

Recitation 26

Adding the AND special form

Now let's add support for AND to the explicit control evaluator. For simplicity, we'll only worry about ANDs of two expressions, like this:

```
(and e1 e2)
```

We'll assume that we have primitive operators first-conjunct and second-conjunct that pull out e1 and e2, respectively, from an AND expression.
First we'll add a clause to eval-dispatch:

```
eval-dispatch
  ...
  (test (op and?) (reg exp))
  (branch (label ev-and))
  ...
```

1. Here is the code for `ev-and`. Fill in the blanks.

```
ev-and

  (assign unev ________________________________________ )

  (assign exp _________________________________________ )

  (save continue)

  (save env)

  (save unev)

  (assign continue eval-after-first)

  (goto _______________________________________________ )

eval-after-first

  (restore ____________________________________________ )
```

```
  (restore __________________________________________ )

  (test (op true?) (reg val))

  (branch (label eval-second-arg))

  (assign val (const #f))

  (restore __________________________________________ )

  (goto (reg continue))

eval-second-arg

  (assign exp _________________________________________ )

  (assign continue eval-after-second)

  (goto _____________________________________________ )

eval-after-second

  _____________________________________________________

  (goto (reg continue))
```

2. Does this `ev-and` routine handle tail recursion? Consider this Scheme code:

```
(define (list? x)
  (or (null? x)
      (and (pair? x) (list? (cdr x)))))

(define z (list 1))
(set-cdr! z z)

(list? z)
```

If you ran this code in MIT Scheme, would you get a stack overflow, or an infinite loop? What if you ran it in the explicit control evaluator using ev-and as implemented above?

3. Make `ev-and` tail-recursive.