January 14, 2014
Keng Low

Short Assignment: The Substitution Model

Trace the evaluation of the expression involving the `mystery` procedure:

`(mystery 7 2)`

- Evaluate `(mystery 7 2)` where `mystery` is defined below.

```
(define mystery
  (lambda ((a <number>) (b <integer>))
    (cond ((zero? b) 0)
          ((odd? b)
           (+ a (mystery (+ a a)
                         (quotient b 2))))
          (else
           (mystery (+ a a)
                    (quotient b 2))))))
```

- Evaluate `lambda` subexpression

`(lambda ((a <number>) (b <integer>))`

Since this is the evaluation of a compound expression, the EVAL rule says we must first evaluate the subexpressions. The value obtained by evaluating the `lambda` subexpression is a prodecure object, denoted:

`[PROC ((x <number>) (b <number>))]`

- Evaluate `cond` expression

`(cond ((zero? b) 0)`

`cond` is a special form, so we handle it specially (has its own rule in tho model).

- Evaluate `((zero? b) 0)`

`(PROC ((b <integer>)) ((zero?) (b)) 0)`

- Evaluate 2

`(PROC ((n <number>)) ((zero?) 2) 0)`

We determine the value of the expression b is 2.

- Apply `zero?` procedure to `2`

`[PROC ((n <number>)) ((zero?) 2) 0]`

We take the body of the procedure, `zero?` and subtitute the corresponding arguments where the parameters appear.

- Evaluate `zero?` procedure

`(zero? 2)`

`zero? x` is a primitive procedure that returns #t if its numeric agreement x is equal to zero, #f otherwise. The above expression returns #f. The consequent is not evaluated.

- Evaluate `cond` expression

`(cond ((odd? b)`

We evaluate `cond`, since the first test yields a false value, we evaluate the next test.

- Evaluate `((odd? b)`

`(PROC ((b <integer>)) ((odd?) b)))`

- Evaulate 2

`(PROC ((b <integer>)) ((odd?) 2)))`

We determine the value of the expression b is 2.

- Apply `odd?` procedure to `2`

`[PROC ((n <number>)) ((odd?) 2) 0]`

- Evaluate `odd?` procedure

`odd? x` returns #t if its integer argument x is odd, #f otherwise. The consequent is not evaluated.

- Evaluate the `cond` expression

We evaluate `cond`, since the first test yields a false value, we evaluate the next test.

- Evaluate the `else` expression

The else clause is optional, but if it is present, and all the test expressions test1 through testn have evaluated to #f, then expr-e is evaluated, and its value becomes the value of the cond expression.

- Evaluate `(mystery (+ a a) (quotient b 2))`

Evaluate `mystery`

- Evaluate `(+ a a)`

```
(PROC (a <number>) (+ a a))
```

- Evaluate 7

```
(PROC (a <number>) (+ 7 7))
```

- Apply procedure

```
[PROC (a <number>) (+ 7 7)]
```

- Evaluate +

```
(+ 7 7)
```

- Apply primitive add

```
[add 7 7]
```

Returns 14.

- Evaluate `(quotient b 2)`

```
(PROC (b <integer>) (quotient b 2))
```

- Evaluate 2

```
(PROC (b <integer>) (quotient 2 2))
```

- Apply `quotient` procedure

```
[PROC (b <integer>) (quotient 2 2)]
```

- Evaluate `(quotient 2 2)`

```
(quotient 2 2)
```

- Apply primitive `quotient`

```
[quotient 2 2]
```

`quotient` returns the quotient of a / b. In other words, if a = qb + r, with $0 \leq r < b$, then (quotient a b) returns q. The above expression returns 1.0.

- Substitute the body of the procedure with values {14} and {1.0}.

```
(mystery 14 1.0)
```

- The process repeats, and so on and so forth.

`(cond ((zero? b) 0)` returns #f. We evaluate the next test.
`((odd? b)` returns #t. We evaluate the consequent.