CS 550: Programming Languages
Midterm

Dustin Ingram

May 10, 2012

1 Functional Programming and Streams in Scheme

1. **Solution:**

```
(define (transpose A)
  (if (null? (car A))
      '()
      (cons (map car A) (transpose (map cdr A)))))
```

2. **Solution:**

```
(define (transpose A)
  (accumulate-n cons-stream the-empty-stream A))
```

2 Substitution Model of Computation and Lambda Calculus

1. **Solution:**

```
(sumf (nth-power 3) 2)
```

becomes:

```
(+ ((nth-power 3) 2) (+ ((nth-power 3) 1) 0))
```

substituting `nth-power` for it's procedure:

```
(+ ((lambda (x) (power x 3)) 2) (+ ((lambda (x) (power x 3)) 1) 0))
```

substituting the operands for the parameters:

```
(+ (power 2 3) (+ (power 1 3) 0))
```

then becomes:

```
(+ 8 (+ 1 0))
```

then:

```
(+ 8 1)
```

finally:

```
9
```

2. **Solution:** The base case is that `(mulc2 zero n) = 0`. In this case, this becomes:

```
(lambda (m n) ((zero (lambda (a) (addc n a))) zero))
```

which becomes:

```
(lambda (m n) ((zero (addc n zero))))
```

becomes:

```
(lambda (m n) ((zero n))))
```

which is:

```
0
```

The inductive step is to consider `mult2` when `m = one`:

```
(lambda (m n) ((one (lambda (a) (addc n a))) zero))
```

which becomes:

```
(lambda (m n) ((one (addc n zero))))
```

becomes:

```
(lambda (m n) ((one n))))
```

which is:

```
n
```

Thus, we know that `(mulc2 m n)` returns the product $m \times n$.

3 Scheme Interpreter

1. **Solution:**

```
((lambda (x y) (+ x y)) 1 2)
```

2. **Solution:** We add the following condition in `eval`:

```
((let? exp) (eval (eval-let exp) env))
```

And we add the following predicate to support this (in the SICP style):

```
(define (let? exp) (tagged-list? exp 'let))
```

The definition for `eval-let` is as follows:

```
(define (eval-let exp)
  (transform (cadr exp) (cddr exp))
  env)
```

The transformer is as follows:

```
(define (transform vars body)
  (cons (make-lambda (assign cadar vars) body) (assign caar vars)))
```

Which uses the following accessor function:

```
(define (assign f vars)
  (if (null? vars)
      '()
      (cons (f vars) (assign f (cdr vars)))))
```

4 Logic Programming and Unification in Prolog

1. **Solution:** When `?- insert(a,[b,c],Z).` is entered into Prolog, you get:

   ```
   Z = [a,b,c] ?
   ```

   To get all possible values, enter `a`:

   ```
   | ?- insert(a,[b,c],Z).

   Z = [a,b,c] ? a

   Z = [b,a,c]

   Z = [b,c,a]

   no
   ```

2. **Solution:** Unification is used to bind variables to values. Here, `a` is being unified with `X`, etc. Backtracking is used to determine if a given goal has a match. If it matches, it continues with the next goal, but if it doesn't, it steps back to the previous goal, releases whatever values it might have obtained, and tries to match that goal again.

3. **Solution:** Here, `outranked-by` is being evaluated before `superior`. This causes the rule to be re-evaluated when the first part of the `and` statement is evaluated, causing an infinite loop.

4. **Solution:** Here, this query results in a match for every pair of supervisors and employees. Since "Oliver Warbucks" supervises four other supervisors, he is listed four times. "Ben Bitdiddle" only supervises one other supervisor, so the rule only matches him once.