



COMPUTER SCIENCE 312 (Fall Term 2009)  
**Programming Language Design**  
Prof. Levy

## Problem Set 2

Due Wednesday 14 October

### 1 Reading Assignment: *Essentials of Programming Languages*, Chapter 1

### 2 Programming Assignment

1. Write a procedure `free-vars` that takes a list structure representing an expression in the lambda calculus syntax (shown at the bottom of page 18) and returns a set (a list without duplicates) of all the variables that occur free in the expression. Then write a procedure `bound-vars` along the same lines. *Hints*: Use the `occurs-free?` predicate on page 19, and write a similar predicate `occurs-bound?`, to support `free-vars` and `bound-vars`. The `makeset` function on page 112 of *The Little Schemer* will help, along with a `filter` function that takes a predicate and a list and returns the list elements that satisfy the predicate: e.g.

```
> (filter odd? '(1 2 3 4 5 6 7 8 9))  
(1 3 5 7 9)
```

2., Extend `occurs-free?` and `occurs-bound?` to handle `if` expressions.

3. Consider the subset of Scheme consisting of (1) identifiers (variables); (2) `if` expressions; (3) `lambda` expressions; and (4) function applications. Write a procedure `lexical-address` that takes an expression from this simplified language and returns the expression with every variable  $v$  replaced by a list of the form  $(v: d p)$ , if  $v$  is a bound variable, or by  $(v: \text{free})$  otherwise. For example:

*over...*

```
> (lexical-address '(lambda (a b c)
                    (if (eqv? b c)
                        ((lambda (c)
                           (cons a c))
                            a)
                        b)))
```

```
(lambda (a b c)
  (if ((eqv?: free) (b: 0 1) (c: 0 2))
      ((lambda (c)
         ((cons: free) (a: 1 0) (c: 0 0)))
        (a: 0 0))
      (b: 0 1)))
```

*Hint:* Represent bound variables as a list of lists, initially empty, of variables that get declared by `lambda`. As you recur, add another list to the big list. For example,

```
(lambda (a b c)
  ...
  (lambda (d e)
```

can be represented by the list `((d e) (a b c))`.