Wellesley College ◊ CS251 Programming Languages ◊ Spring, 2000

PROBLEM SET 4
Due Friday, March 31, 2000

Reading

- Environment model: *SICP* 3.2 – 3.2.2
- Interpretation of a higher-order functional language (i.e., Scheme, which is pretty much the same as HOFL): *SICP* 4 – 4.2.1.

Notes:

- You might want to get a head start on reading about the ML programming language, which will begin discussing after Spring break*: Paulson's ML for the Working Programmer (MLWP)*, Chapters 2, 3, 4, 5.1-5.11, 9.

- This problem set is worth 130 points. There are also two extra credit problems worth 50 points.

- The FOFL, FOBS, and HOFL languages mentioned in the problem set were all introduced in class. Implementations exist for each of these three languages. To experiment with the interpreters for these languages, cd to the directory /usr/users/cs251/download/ZZZ and evaluate `(load "load-ZZZ.scm")`, where *ZZZ* ranges over `fofl`, `fobs`, and `hofl`. In FOFL, you can run your programs with any of `env-eval-global`, `env-eval-flat`, or `env-eval-dynamic`. In FOBS and HOFL, you can run your programs with either of `env-eval-static` or `env-eval-dynamic`.

Problem 1 [20]: Block Structure

It is possible to translate any program in a first-order block-structured language into a first-order language without block structure. This can be done by "lifting" all function declarations to top-level, possibly renaming some of the functions and adding some extra formal and actual parameters to some functions.

Illustrate this fact by manually translating the FOBS (first-order, block structured) program in Figure 1 (on the next page) into a FOFL (first-order, no block-structure) program that has the same meaning. Your FOFL program should have exactly the same number of function declarations as the FOBS program, but all of the FOFL declarations should be top-level. You should assume that the FOBS program is statically scoped and the FOFL program is globally scoped.

Notes: The program in Figure 1 appears as the program `pythagorean-triples` in the FOBS test suite in the file `fobs-examples.scm` as within the `fobs` directory of the CS251 download directory. After you have loaded FOBS (by evaluating `(load "load-fobs.scm")`), you can test the program by evaluating expressions like:

```
(env-run-static pythagorean-triples '(5))
```

Note that evaluating this program can take a long time even for a small input. The expected result for an input of 5 is `((4 3 5) (3 4 5))`.

You are encouraged to extend the test suite with your solution to this problem to test it and make sure it works.

```
(program (n)
  (funrec ; #1
    ((sq (x) (* x x))
     (triple (x y z) (prepend x (prepend y (prepend z (empty)))))
     (to (hi)
       (funrec ; #2
         ((loop1 (x)
            (if (> x hi)
                (empty)
                (prepend x (loop1 (+ x 1))))))
         ;; Body of funrec #2
         (loop1 1)))
     (triples (lst1 lst2 lst3)
       (funrec ; #3
         ((loop1 (L1 ans1)
            (if (empty? L1)
                ans1
                (bind n1 (head L1)
                  (funrec ; #4
                    ((loop2 (L2 ans2)
                       (if (empty? L2)
                           (loop1 (tail L1) ans2)
                           (bind n2 (head L2)
                             (funrec ; #5
                               ((loop3 (L3 ans3)
                                  (if (empty? L3)
                                      (loop2 (tail L2) ans3)
                                      (bind n3 (head L3)
                                        (if (= (+ (sq n1) (sq n2))
                                               (sq n3))
                                            (loop2 (tail L2)
                                                   (prepend (triple n1 n2 n3)
                                                            ans3))
                                            (loop3 (tail L3) ans3))))))
                                ;; Body of funrec #5
                                (loop3 lst3 ans2))))))
                    ;; Body of funrec #4
                    (loop2 lst2 ans1))))))
         ;; Body of funrec #3
         (loop1 lst1 (empty)))))
    ;; Body of funrec #1
    (triples (to n) (to n) (to n))))
```

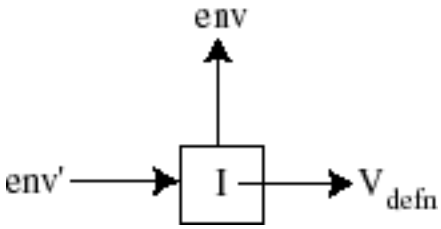
Figure 1: A FOBS program computing Pythagorean triples (a b c), where a, b, c are all integers in the range [1..n]. A triple (a b c) is a Pythagorean triple if $a^2 + b^2 = c^2$.

Problem 2 [20]: Static Scope

Consider the following HOFL program:

```
(program (a)
  (bind linear (abs (a b)
                  (abs (x)
                    (+ (* a x) b)))
    (bind line1 (linear 1 2)
      (bind line2 (linear 3 4)
        (bind try (abs (b)
                    (prepend (line1 b)
                             (prepend (line2 (+ b 1))
                                      (prepend (line2 (+ b 2))
                                               (empty)))))
          (try (+ a a)))))))
```

a. Suppose that the above program is run on the input list '(5). Draw an environment diagram that shows all of the environments and closures that are during the evaluation of this program in statically scoped HOFL. In order to simplify the diagram , you should treat `bind` as if it is a kernel construct and ignore the fact that it desugars into an application of a `abs`. That is, you should treat the evaluation of (`bind` *I* E_{defn} E_{body}) in environment *env* as the result of evaluating E_{body} in the environment frame *env'* shown below (where V_{defn} is the result of evaluating E_{defn} in *env*.):



b. What is the value of the above expression in staticallytscoped HOFL? (You should figure out the answer on your own, but may wish to check it using the statically scoped HOFL interpreter. To load the interpreter, evaluate (`load "load-hofl.scm"`). To evaluate a program, use (`env-run-static` *`program arguments`*).)

Problem 3 [20]: Dynamic Scope

a. Draw an environment diagram that shows all of the environments created during the evaluation of the program from Problem 2 in dynamically scoped HOFL.

b. What is the value of evaluating the program from Problem 2 in dynamically scoped HOFL? (You should figure out the answer on your own, but may wish to check it using the dynamically-scoped HOFL interpreter in the `hofl` directory of the CS251 download directory. To load the interpreter, evaluate (`load "load-hofl.scm"`). To evaluate a program, use (`env-run-dynamic` *`program arguments`*).)

c. In a programming language with higher-order functions, which supports modularity better: lexical scope or dynamic scope? Explain your answer.

Problem 4 [15]: Variable and Function Scoping

As discussed in lecture, the first-order block-structured language FOBS has two namespaces: one for functions and one for variables. It is possible to independently choose between static and

dynamic scoping in each of the two namespaces by tweaking the definition of `funapply` within the FOBS environment model interpreter:

```
(define funapply
  (lambda (closure actuals fenv venv)
    (env-eval-static (closure-body closure)
                     function-env
                     (env-extend (closure-params closure)
                                 actuals
                                 variable-env
                                 ))))
```

The choice of scoping for the function namespace is determined by the expression *`function-env`*. If this expression is `(closure-fenv closure)`, then function names have static scope; if this expression is `fenv`, then function names have dynamic scope. Similarly, the choice of scoping for the variable namespace is determined by the expression *`variable-env`*. If this expression is `(closure-venv closure)`, then variable names have static scope; if this expression is `venv`, then variable names have dynamic scope.

In this problem you are to write a single FOBS program whose value indicates which scoping mechanisms are being used for function names and variable names. We will assume that supports string literals in addition to integer and boolean literals. For example, here is a simple FOBS program that returns a string classifying its integer input:

```
(program (a) (if (< a 0) "negative" (if (= a 0) "zero" "positive")))
```

Your FOBS program should take zero arguments, and return a two-element list of strings (*`fstring vstring`*), where

- *`fstring`* is `"statfun"` if function names are statically scoped and `"dynfun"` if they are statically scoped.
- *`vstring`* is `"statvar"` if variable names are statically scoped and `"dynvar"` if they are dynamically scoped.

The only types of values that your program should manipulate are strings, functions, and lists of strings. That is, your example should not involve any integers or booleans. Strive to make your program as simple and understandable as possible. You can test out your program by tweaking the environment model interpreter in the `fobs` subdirectory of the CS251 course directory.

Note: If you cannot solve this problem with just strings, functions, and list of strings, you can get partial credit by solving the problem using other types of values.

Problem 5 [15]: Bindrec

Consider the following HOFL expression *E*:

```
(bindseq ((f (abs (x) (+ x 1))))
  (bindrec ((f (abs (n)
                 (if (= n 0)
                     1
                     (* n (f (- n 1)))))))
    (f 3)))
```

a. What is the value of *E* in lexically scoped HOFL?

b. Consider the expression *E′* that is obtained from *E* by replacing the underlined `bindrec` by `bindseq`. What is the value of *E′* in lexically scoped HOFL?

c. What is the value of the expression *E′* from part B in dynamically scoped HOFL?

d. Does a dynamically scoped language need a recursive binding construct like `bindrec` in order to support the creation of local recursive procedures?

PROBLEM 6 [40]: Recur

Local recursive functions defined via HOFL's `bindrec` or Scheme's `letrec` are often hard to read. Here is a more readable construct for simple local recursions:

`(recur` I_{name} `((`I_1 E_1`) ... (`I_n E_n`))` E_{body}`)`
Create a local function named I_{name} whose formal parameters are I_1 through I_n and whose body is E_{body}. The value returned by `recur` is the result of applying the local function to the values that result from evaluating the initial value expressions E_1 through E_n. E_{body} is in the scope of I_{name} as well as I_1 through I_n, so I_{name} may be recursive. However, E_1 through E_n are not in the scope of I_{name} or I_1 through I_n.

For example, here is a version of factorial that uses a local `loop` function introduced by `recur`:

```
(define fact
  (lambda (n)
    (recur loop ((num n) (ans 1))
      (if (= num 0)
          ans
          (loop (- num 1) (* num ans))))))
```

In this problem, you will be extending the HOFL implementation to support the `recur` construct. In your extensions, you should use the following abstract syntax operations for the `recur` construct, which have already been implemented for you in `abstract-syntax.scm`.

```
(make-recur name formals inits body)
(recur-name recur-exp)  ; I_name
(recur-formals recur-exp) ; I_1 through I_n
(recur-inits recur-exp) ; E_1 through E_n
(recur-body recur-exp) ; E_body
(recur? s-expression)
```

The HOFL implementation is in the `hofl` subdirectory of the CS251 download directory. You should make a local copy of this directory so that you can edit the files. You should turn in the following three files after solving the problems below: `desugar.scm`, `free-variables.scm`, `env-eval-static.scm`. To load the HOFL system, evaluate `(load "load-hofl.scm")`.

a [10]. Based on the above specification of the `recur` construct, extend the definition of the free variables function `free-vars` in `free-vars.scm` to handle the `recur` construct. Test your definition by evaluating `(test-recur-free-vars free-vars)`.

b [15]. Add a clause to the `env-eval-static` function in `env-eval-static.scm` that evaluates the `recur` construct according to the above specification. Make sure you understand the bindrec clause before you attempt to implement the recur clause. Test your extension by evaluating `(test-recur env-run-static)`.

c [15]. An alternative to extending the `env-eval-static` function to handle `recur` is to instead modify the `desugar` function in `desugar.scm` to desugar recur into other HOFL constructs. You should use `bindrec` in your desugaring. There is already a "stub" clause for recur in `desugar.scm` that just desugars the parts of a `recur` but leave the recur in place. You should replace this stub with a clause that removes the `recur` altogether. Test your extension by evaluating `(test-recur env-run-static)`.

Extra Credit 1 [10] FOFL Scoping

In class, we discussed three kinds of scoping for variable names in the FOFL language: flat, global, and dynamic. Suppose that FOFL supports string literals (like FOBS in Problem 4). By analogy with Problem 4, we would like to write a zero-argument program that, when executed, returns "flat" if FOFL variable scoping is flat, "global" if FOFL variable scoping is global, and "dynamic" if FOF L variable scoping is dynamic.

Is it possible to write such a program? If yes, write the program. If no, explain why it cannot be written.

Extra Credit 2 [40] Automatically translating FOBS to FOFL

In Problem 1, you manually translated a FOBS program into a FOFL program. It is possible to write a program that automatically translates any FOBS program into a FOFL program that has the same meaaning. Write this translation program.

Partial credit will be awarded for any progress on this problem.

Problem Set Header Page:
Please make this the first page of your submission.

CS251 Problem Set 4
Due Friday, March 31, 2000

Name:

Date & Time Submitted *(only if late)*:

Collaborators (*anyone you collaborated with in the process of doing the problem set)*:

*In the **Time** column, please estimate the time you spent on the parts of this problem set. Please try to be as accurate as possible; this information will help me to design future problem sets. I will fill out the **Score** column when grading your problem set.*

Part	Time	Score
General Reading		
Problem 1 [20]		
Problem 2 [20]		
Problem 3 [20]		
Problem 4 [15]		
Problem 5 [15]		
Problem 6a [10]		
Problem 6b [15]		
Problem 6c [15]		
Extra Credit 1 [10]		
Extra Credit 2 [40]		
Total [130]		