PPL242

Assignment 2

Responsible Lecturer: Meni Adler
Responsible TA: Brahan Wassan

Submission Date: 20/6/2024

General Instructions

Submit your answers to the theoretical questions in a pdf file called id1_id2.pdf and your code (the L3 folder), and ZIP those files together into a file called *id1_id2.zip*.
Do not send assignment related questions by e-mail, use the forum instead. For any administrative issues (milu'im/extensions/etc) please open a request ticket in the Student Requests system.

You are provided with the templates *ex2.zip*.
Unpack the template files inside a folder. From the command line in that folder, invoke `npm install`, and work on the files in that directory, preferably working in the Visual Studio Code IDE (refer to the Useful Links). In order to run the tests, run `npm test` from the command line.

Important: Do not add any extra libraries and do not change the provided package.json and tsconfig.json configuration files. **The graders will use the exact provided files**. If you find any missing necessary libraries, please let us know.

Question 1: Theoretical Questions [30 points]

1.1 Is a function-body with multiple expressions required in a pure functional programming? In which type of languages is it useful? What about L3? [2 points]

Q1.2

a. Why are special forms required in programming languages? Why can't we simply define them as primitive operators? Give an example [3 points]
b. Can the logical operation 'or' be defined as a primitive operator, or must it be defined as a special form? Refer in your answer to the option of shortcut semantics. [3 points]

Q1.3

a. What is the value of the following L3 program? Explain. [2 points]

```
(define x 1)
(let ((x 5)
      (y (* x 3)))
  y)
```

b. Read about let* here.
   What is the value of the following program? Explain. [2 points]

```
(define x 1)
(let* ((x 5)
       (y (* x 3)))
  y)
```

c. Define the let* expression above as an equivalent let expression [2 points]
d. Define the let* expression above as an equivalent application expression (with no let) [2 points]

Q.4 [6 points]

a. In L3, what is the role of the function valueToLitExp?
b. The valueToLitExp function is not needed in the normal evaluation strategy interpreter (L3-normal.ts). Why?
c. The valueToLitExp function is not needed in the environment-model interpreter. Why?

Q.5 [4 points]

a. What are the reasons that would justify switching from applicative order to normal order evaluation? Give an example.
b. What are the reasons that would justify switching from normal order to applicative order evaluation? Give an example.

Q.6 [4 points]

a. Why is renaming not required in the environment model?
b. Is renaming required in the substitution model for a case.where the term that is substituted is "closed", *i.e.*, does not contain free variables? explain.

Answers should be submitted in file id1_id2.pdf

Question 2: Adding Class to L3 [70 points]

The 'L3' directory in the assignment template contains the parser of L3 and the interpreter for both substitution and environment models.'

In this question we extend L3 with ‘class’ special form.
A ‘class’ expression is defined by a list of ‘fields’ and a list of methods (as bindings - method names and method expressions), as given by the following abstract and concrete syntax:

```
<program> ::= (L31 <exp>+)              / Program(exps:List(exp))
<exp> ::= <define> | <cexp>             / DefExp | CExp
<define> ::= ( define <var> <cexp> )    / DefExp(var:VarDecl,
val:CExp)
<var> ::= <identifier>                  / VarRef(var:string)
<cexp> ::= <number>                     / NumExp(val:number)
        |  <boolean>                    / BoolExp(val:boolean)
        |  <string>                     / StrExp(val:string)
        |  ( lambda ( <var>* ) <cexp>+ ) / ProcExp(args:VarDecl[],
        |                                /         body:CExp[]))
        |  ( class ( <var>+ ) ( <binding>+ ) )
                  / ClassExp(fields:VarDecl[], methods:Binding[]))
        |  ( if <cexp> <cexp> <cexp> ) / IfExp(test: CExp,
        |                                      then: CExp,
        |                                      alt: CExp)
        |  ( let ( <binding>* ) <cexp>+ ) /
LetExp(bindings:Binding[],
        |                                      body:CExp[]))
        |  ( quote <sexp> )                / LitExp(val:SExp)
        |  ( <cexp> <cexp>* )               / AppExp(operator:CExp,
        |                                         operands:CExp[]))
<binding>  ::= ( <var> <cexp> )            / Binding(var:VarDecl,
        |                                           val:Cexp)
<prim-op>  ::= + | - | * | / | < | > | = | not |  eq? | string=?
                 | cons | car | cdr | list | pair? | list? | number?
                 | boolean? | symbol? | string?
<num-exp>  ::= a number token
<bool-exp> ::= #t | #f
<str-exp>  ::= "tokens*"
<var-ref>  ::= an identifier token
<var-decl> ::= an identifier token
<sexp>     ::= symbol | number | bool | string | ( <sexp>* )
```

The class Pair in Java, for example, can be defined in L31 with the new special for as follows:

```
// Java
class Pair {
    private int a,b;
```

```
    Pair(int a, int b) { this.a = a; this.b = b; }
    int first() { return a; }
    int second() { return b; }
    int sum() { return a + b; }
    Pair scale(k)  { return new Pair(k*a, k*b); }
}

//L31
(define pair
    (class (a b)
         ((first (lambda () a))
          (second (lambda () b))
          (sum (lambda () (+ a b)))
          (scale (lambda (k) (pair (* k a) (* k b))))
         )
    )
)
```

a. Add the new 'class' special form to the parser of L3 (the file 'L3/L3-ast.ts') [15 points]

   You can test your code with test/q2a.tests.ts

   [15 points]

b. Add the semantics of the new *class* special form to the interpreter, for both substitutional model (L3/L3-eval-sub.ts and L3/L3-value.ts files) and environment model (L3/L3-eval-env.ts and L3/L3-value.ts files):

   - The value of ClassExp is a Class value (you can define it as you wish)

     ```
     pair
     → Class
     ```

   - In order to create an instance of a given class, the class should be 'applied' with the parameters for the fields (as done with constructor in languages like Java)

     ```
     // Java
     Pair p34 = new Pair(3,4);

     // L31
     (define p34 (pair 3 4))
     ```

The value of ‘instance’ of ClassExp is a **Object** value (you can define it as you wish)

```
p34
→ Object
```

- In order to ‘call’ method, the object should be applied with a symbol which denotes the method:

```
// Java
p34.first();
→ 3
p43.second();
→ 4
p34.sum();
→ 7
p34.scale(2).second();
→ 8

// L31
(p34 'first)
→ 3
(p34 'second)
→ 4
(p34 'add)
→ 7
((p34 'scale 2) 'second)
→ 8

(p34 'power)
→ {tag: 'Failure', message: 'Unrecognized method: power'}
```

You can test your code with test/q2b.sub.tests.ts and test/q2b.env.tests.ts

[30 points]

c. A given ‘class’ expression can be transformed to an equivalent AppExp (so we can avoid the interpreter modifications of 2.b).

The Pair class, for example, can be expressed by the following ProcExp (**for simplicity, in this section we assume that the methods of the class have no parameter, and that #f indicates error in the transformed code as done in the following example**):

```
(define pair
  (lambda (a b)
    (lambda (msg)
      (if (eq? msg 'first)
          ((lambda () a) )
          (if (eq? msg 'second)
              ((lambda () b) )
              (if (eq? msg 'sum)
                  ((lambda () (+ a b)) )
                  #f))))))
```

Implement the procedure *class2proc* (at file L3/LexicalTransformations.ts), which applies a syntactic transformation from a ClassExp to an equivalent ProcExp.

Implement the procedure *lexTransform* (at file L3/LexicalTransformations.ts), which gets a program with *class* forms and returns an equivalent program with no *class* forms.

You can test your code with test/q2c.tests.ts

[15 points]

d. Given the following program:

```
(define pi 3.14)
(define square (lambda (x) * x x))
(define circle
    (class (x y radius)
         (
          (area (lambda () (* (square radius) pi)))
          (perimeter (lambda () (* 2 pi radius)))
         )
    )
)
(define c (circle 0 0 3))
(c 'area)
```

- Convert the ClassExps to ProcExps (as defined in 2.c)

- List the expressions which are passed as operands to the L3applicativeEval function during the computation of the program, (after the conversion), for the case of substitution model.

-

For example:

```
(define x 3)
(+ x 3)
→
3
(+ x 3)
+
x
3
```

- Draw the environment diagram for the computation of the program (after the conversion), for the case of the environment model interpreter.

Answers should be submitted in file id1_id2.pdf

[10 points]

Good Luck!