PPL212

Assignment 2

Responsible Lecturer: Meni Adler
Responsible TA: Or Hayat

Submission Date: 18/4/2021

General Instructions

Submit your answers to the theoretical questions in a pdf file called id1_id2.pdf and your code for programming questions inside the provided q2.l3, and L31-ast.ts, q3.ts, q4.ts files of the *src* folder. ZIP those files together (including the pdf file, and only those files) into a file called *id1_id2.zip*. Make sure that your code abides the Design By Contract methodology.
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]

Q1.1 Why are special forms required in programming languages? Why can't we simply define them as primitive operators? Give an example [4 points]

Q1.2 Write a program in L1, containing more than one expression, where the evaluation of the program's expression **can be done in parallel** (e.g., the interpreter can run a thread for each expression evaluation).
Write a program in L1, containing more than one expression, where the evaluation of the program's expression **cannot be done in parallel**.
[4 points]

Q1.3 Let us define the L0 language as L1 excluding the special form ‘define’. Is there a program in L1 which cannot be transformed to an equivalent program in L0? Explain or give a contradictory example [4 points]

Q1.4 Let us define the L20 language as L2 excluding the special form ‘define’. Is there a program in L2 which cannot be transformed to an equivalent program in L20? Explain or give a contradictory example [4 points]

Q1.5 For the following high-order functions in L3, which get a function and a list, indicate (and explain) whether the order of the procedure application on the list items should be sequential or can be applied in parallel:

- map
- reduce
- filter
- all (returns #t is the application of the given boolean function on each of the given list items returns #t)
- compose (compose a given procedure with a given list of procedures)

[10 points]

Q1.6 Regarding L31 language, as defined in **Q3b** (below): what is the value of the following program? Explain.

```
(define b 1)
(define c 2)
(define pair
    (class (a b)
         ((first (lambda () a))
          (second (lambda () b))
          (f (lambda () (+ a b c)))
    )
)

(define p34 (pair 3 4))

(
   (lambda (c) (p34 'f))
   5
)
```

[4 points]

<u>Answers should be submitted in file id1_id2.pdf</u>

Question 2: Programing in L3 [30 points]

Q2.1

Write an L3 procedure *append*, which gets two lists and returns their concatenation. For example:
(append '(1 2) '(3 4)) → '(1 2 3 4)

Q2.2

Write an L3 procedure *reverse*, which gets a list and reverses it. For example:
(reverse '(1 2 3)) → '(3 2 1)

Q2.3

Write an L3 procedure *duplicate-items*, which gets two lists - *lst* , *dup-count* - and duplicates each item of *lst* according to the number defined in the same position in *dup-count*.
In case *dups-count* length is smaller than *lst*, *dup-count* should be treated as a cyclic list.
Examples:

(duplicate-items '(1 2 3) '(1 0))→ '(1 3)
(duplicate-items '(1 2 3) '(2 1 0 10 2))→ '(1 1 2)

You may assume that dup-count contains numbers and is not empty.

Q2.4

Write an L3 procedure *payment*, which gets a sum of money and list of available coins, and returns the number of possible ways to pay the money with these coins.
Examples:

(payment 10 ‘(5 5 10)) → 2
[1 coin of 10 ,2 coins of 5]

(payment 5 ‘(1 1 1 2 2 5 10) → 3
[1 coin of 5, 1 coin of 2 and 3 coins of 1, 2 coins of 2 and 1 coin of 1]

Q2.5

Write an L3 procedure *compose-n*, which gets an unary function *f* and a number *n* (>0) and returns the closure of the n-th self-composition of f:
For example:

(define mul8 (compose-n (lambda (x) (* 2 x)) 3))
(mul8 3) → 24

You may add auxiliary procedures to all questions.

<u>The code **(without comments)** should be submitted in file src/q2.l3</u>

Don't forget to write a contract for each of the above procedures.

```
; Signature:
; Type:
; Purpose:
; Pre-conditions:
; Tests:
```

<u>Write the contracts in file id1_id2.pdf.</u>

You can test your code with test/q2-tests.ts

Question 3: Syntactic Parsing & Transformations [25 points]

Let us define the L31 as L3 with the addition of the special form 'class' for class definition:
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; }
}

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

a. Add the new ‘class’ special form to the parser of L31 (the file ‘src/L31-ast.ts’):

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))
```

In order to call a method in L31, the instance should be applied with a symbol which denote the method:

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

// L31
(p34 ‘first)
→ 3
(p34 ‘second)
→ 4
(p34 ‘add)
→ 7
```

In order to avoid the implementation of the semantics of the ‘class’ form in the interpreter of L31, one of the students suggested transforming a given ‘class’ expression to an equivalent AppExp. The Pair class, for example, can be expressed by the following ProcExp (**for simplicity, 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))))))
```

b. Implement the procedure *class2proc* (at file src/q3.ts), which applies a syntactic transformation from a ClassExp to an equivalent ProcExp.

   Implement the procedure *L31ToL3* (at file src/q3.ts), which gets an L31 AST and returns an equivalent L3 AST.

<u>The code should be submitted in files src/q3.ts, src/L31-ast.ts</u>
You can test your code with test/q3-tests.ts

Question 4: Code translation [15 points]

Write the procedure *l2ToPython* which transforms a given L2 program to a Python program.

For example:

```
(+ 3 5) ⇒ (3 + 5)

(if (> x 3) 4 5) ⇒ (4 if (x > 3) else 5)

(lambda (x y) (* x y)) ⇒ (lambda x, y : (x * y))

((lambda (x y) (* x y)) 3 4) ⇒  (lambda x, y : (x * y))(3,4)

(define pi 3.14) ⇒ pi = 3.14

(define f (lambda (x y) (* x y))) ⇒ f = (lambda x, y : (x * y))

(f 3 4) ⇒ f(3,4)

boolean? ⇒ (lambda x : (type(x) == bool))

(L3
(define b (> 3 4))
(define x 5)
(define f (lambda (y) (+ x y)))
(define g (lambda (y) (* x y)))
(if (not b) (f 3) (g 4))
(if (= a b) (f 3) (g 4))
```

```
(if (> a b) (f 3) (g 4))
((lambda (x) (* x x)) 7)
)
⇒
b = (3 > 4)
x = 5
f = (lambda y : (x + y))
g = (lambda y : (x * y))
(f(3) if (not b) else g(4))
(f(3) if (a == b) else g(4))
(f(3) if (a > b) else g(4))
(lambda x : (x * x))(7)
```

The procedure gets an L2 AST and returns a string of the equivalent Python program.

To make things simpler, you can assume that the body of the lambda expressions contains <u>one</u> expression.

<u>Note</u>: The primitive operators of L2 are: +, -, *, /, <, >, =, number?, boolean?, eq?, and, or, not

<u>Hint</u>: Take a look at the unparse procedure.

<u>The code should be submitted in file src/q4.ts</u>
You can test your code with test/q4-tests.ts

Good Luck!