PPL192

Assignment 2

Responsible Lecturer: Meni Adler
Responsible TA: Roni Zoller

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, q3.ts, q4.ts files in the correct places. 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.
Important: do not add any extra libraries in the supplied template files, otherwise, we will fail to compile and you will receive a grade of zero. If you find that we forgot to import necessary libraries, let us know.

Question 1: Theoretical Questions [30 points]

Q1.1 Give an example for each of the following categories in L3:

- Primitive atomic expression
- Non-primitive atomic expression
- Non-primitive compound expression
- Primitive atomic value
- Non-primitive atomic value
- Non-primitive compound value

[6 points]

Q1.2 What is a special form? Give an example [2 points]

Q1.3 What is a free variable? Give an example [2 points]

Q1.4 What is Symbolic-Expression (s-exp)? Give an example [2 points]

Q1.5 Give two examples for syntactic abbreviation [4 points]

Q1.6 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]

Q1.7 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]

Q1.8 In practical session 5, we dealt with two representations of primitive operations: *PrimOp* and *Closure*. List an advantage for each of the two methods [2 points].

Q1.9 In class, we implemented *map* in L3, where the given procedure is applied on the first item of the given list, then on the second item, and so on. Would another implementation which applies the procedure in the opposite order (from the last item to the first one), while keeping the original order of the items in the returned list, be equivalent? Would this be the case also for *reduce* [4 points]

Answers should be submitted in file id1_id2.pdf

Question 2: Programing in L3 [30 points]

Q2.1 Implement in L3 the procedure *empty?*, which returns true iff the given expression is the empty list. For example:
(empty? ‘()) ⇒ #t
(empty? ‘(1 2 3)) ⇒ #f
(empty? 4) ⇒ #f
[2 points]

Q2.2 Implement in L3 the procedure *list?*, which returns true iff the given expression is a list (according to the inductive definition presented in class). For example:
(list? ‘()) ⇒ #t
(list? (cons 1 (cons 2 ‘()))) ⇒ #t
(list? (cons 1 2) ⇒ #f
[4 points]

Q2.3 Implement in L3 the procedure *equal-list?*, which returns true iff the given two expressions are equal lists. For example:
(equal-list? ‘(1 2) ‘(1 2)) ⇒ #t
(equal-list? ‘(1 2) ‘(1 3)) ⇒ #f
(equal-list? '(#t "a" (2 'b)) '(#t "a" (2 'b)))) ⇒ #t

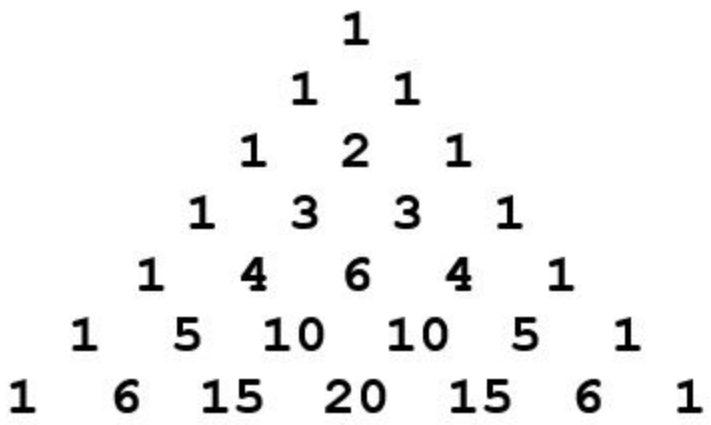
(equal-list? (cons 1 2) (cons 1 2)) ⇒ #f
[6 points]

Q2.4 Implement the procedure *append*, which takes two lists and returns the first list appended with the second. For example:
(append '(1 2 3) '(4 5 6)) ⇒ '(1 2 3 4 5 6).
[4 points]

Q2.5 Implement the procedure *append3*, which takes two lists and one item, and returns the first list appended with the second and the integer. For example:
(append3 '(1 2 3) '(4 5 6) 7) ⇒ '(1 2 3 4 5 6 7).
[4 points]

Q2.6 Pascal's Triangle is defined by the following rules:

- The topmost row is 1
- The nth row is computed by adding adjacent numbers in the n-1 row, and adding 1 at the beginning and at the end.



Implement the procedure *pascal*, which takes an integer n, and computes the nth row of Pascal's triangle. (you may use the previous procedures and add one more auxiliary procedure)
For example, (pascal 5) => '(1 4 6 4 1)
[10 points]

The code **(without comments)** should be submitted in file q2.l3

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

```
; Signature:
; Type:
; Purpose:
```

```
; Pre-conditions:
; Tests:
```
Write the contracts in file id1_id2.pdf.

You can test your code with q2-tests.ts

Question 3: Syntactic Transformations [20 points]

Let us define the L30 language as L3 excluding the *list* primitive operation and the literal expression for lists.
In this question you are required to implement a syntactic transformation from L3 to L30.

Implement the procedure *l3ToL30* which gets an L3 AST and returns L30 AST.
Your code should deal with 2 cases:

- Transformation of *list* application to *cons* application or empty list. For example:
(list) ⇒ '()
(list 1 2) ⇒ (cons 1 (cons 2 '())
(list (list 1 2) (list 3 4)) ⇒ (cons (cons 1 (cons 2 '())) (cons (cons 3 (cons 4 '())) '()))

- Transformation of list literal expressions to *cons* application. For example:
'(1 2) ⇒ (cons 1 (cons 2 '())
'('(1 2) '(3 4)) ⇒ (cons (cons 1 (cons 2 '())) (cons (cons 3 (cons 4 '())) '()))

Don't forget to write the contract of the *l3ToL30* procedure as a comment in the code.
The code should be submitted in file q3.ts
You can test your code with q3-tests.ts

Question 4: Code translation [20 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)

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.

Note: As a starting point you may take the *unparse* procedure, which gets an L2 AST and returns an L2 program string.

```
export const unparse = (exp: Parsed | Error): string | Error =>
    isError(exp) ? exp.message :
    isProgram(exp) ? map(unparse,exp.exps).join("\n") :
    isBoolExp(exp) ? (exp.val ? '#t' : '#f') :
    isNumExp(exp) ? exp.val.toString() :
    isVarRef(exp) ? exp.var :
    isPrimOp(exp) ? exp.op :
    isDefineExp(exp) ? "(define " + exp.var.var + " " +
                                    unparse(exp.val) + ")" :
    isProcExp(exp) ? "(" + "lambda (" +
                          map((p) => p.var, exp.args).join(" ") + ") " +
                          map(unparse, exp.body).join(" ") +
                     ")" :
    isIfExp(exp) ? "(" + "if " +
                         unparse(exp.test) + " " +
                         unparse(exp.then) + " " +
                         unparse(exp.alt) +
                   ")" :
    isAppExp(exp) ? "(" +
                          unparse(exp.rator) + " " +
                          map(unparse, exp.rands).join(" ") +
                    ")" :
    Error("Unknown expression: " + exp.tag);
```

Don’t forget to write a contract for the *l2ToPython* procedure, a a comment in the code.
<u>The code should be submitted in file q4.ts</u>

You can test your code with q4-tests.ts