CMPUT 325: Non-Procedural Programming Languages

Introduction to Assignment 2

Md Solimul Chowdhury
mdsolimu@ualberta.ca

Assignment 2

- LISP and Functional Programming
  - 15 marks
- Two parts:
  - Part I : Programming in LISP
    - 11 marks
    - Due on Sunday, February 28th, 11:55 pm.
  - Part II: A Quiz on Functional Programming
    - 4 marks
    - Due on Monday, March 1st, 11:55 pm.

Part I: The Programming Assignment

- Task: is to implement an Interpreter in LISP
  - for a given functional language named FL
  - Interpreter: itself a program that executes a given program.
- FL is similar to the toy language FUN discussed in the class.

FL-INTERP: THE INTERPRETER FOR FL

- Implement a function **fl-interp**
  - Signature of fl-interp:

    (fl-interp E P)

    - where, P is given program in FL and E is a valid Expression in FL,
      - Program P is a collection of functions
    - **fl-interp** returns the result of evaluating E with respect to P.

Syntax of FL (1)

- Does not have **defun** to define a function.
- Instead a function is defined in the following way:

f(X1,...,Xn) = Exp

  - **f** is the name of the function you want to define
  - **X1,..,Xn** are the list of parameters that f accepts
  - **Exp** is the expression in FL that implements f
- Example:
  - square (x) = (* x x))
    - f → square
    - X1,...,Xn →x
    - Exp → (* x x)

Syntax of FL (2)

- Restrictions:
  - We consider:
    - **Higher-order functions** that take a function as an input but do not return a function as the output.
  - But, No Lambda Notations.

Syntax of FL (3)

- function application:
  - (f e1 ... en)
  - f is the function name and **e1, …, en** are its arguments
    - concrete values that replaces parameters equals by equals

- Example:
  - P=(xmember (X L) = ...)
  - Function applicaiton: (xmember a (b c d a))
  - (fl-interp '(xmember a (b c d a)) '((xmember (X L) = ...)))

18 Primitieve functions to implement

The following primitive functions must be implemented. The meanings of these functions are the same as those in Lisp, where *first* and *rest* are identified with *car* and *cdr*, respectively.

```
(if x y z)
(null x)
(atom x)
(eq x y)
(first x)
(rest x)
(cons x y)
(equal x y)
(number x)  return T (true) if x is a number, NIL otherwise.
       Same as (numberp x) in Lisp
(+ x y)
(- x y)
(* x y)
(> x y)
(< x y)
(= x y)
(and x y)
(or x y)
(not x)
```

As in Lisp, we use the atom NIL for the truth value false, and **anything else represents true when evaluated** as a boolean expression.

Unlike Lisp, **always return the constants T and NIL for all the boolean functions you implement**, such as equal, =, and, ...

Control flow inside the interpreter (1)

- Evaluation of a function application

  → by replacing equals by equals.

- Example:

  function application: (f e1 ... en)

  function definition: (f (X1 ... Xn) = Body)

  - Replace (f e1 ... en) by Body
  - and in Body, replace each occurrence of

    X1 with e1,

    X2 with e2,

    ….

    Xn with en

Control flow inside the interpreter (1)

- Reduction:
  - Each step of evaluation is called reduction.
  - Performed repeatedly until no more reduction is possible.
  - **<u>Goal of the interpreter</u>**: `to` reduce a given expression to a normal form using the definitions in the given program

- fl-interp should implement ***applicative order reduction***

Control flow inside the interpreter (2)

- *applicative order reduction*
  - the arguments be evaluated before a function is applied
  - applies the (leftmost) innermost function first.

Suppose P is

```
(
  (f (X Y) =  (+ X Y))
  (g (X) =   (+ 1 X))
)
```

Consider applicative order reduction of (f (g 2) (g 1)).

```
   (f (g 2) (g 1))
=> (f (+ 1 2) (g 1))
=> (f 3 (g 1))
=> (f 3 (+ 1 1))
=> (f 3 2)
=> (+ 2 3)
=> 5
```

In contrast, normal order reduction applies the outermost leftmost applicable function first.

```
   (f (g 2) (g 1))
=> (+ (g 2) (g 1)) ; f is applied even if its arguments have not been evaluated
=> (+ (+ 1 2) (g 1))
=> (+ 3 (g 1))
=> (+ 3 (+ 1 1))
=> (+ 3 2)
=> 5
```

Some edge cases

- A function is identified by its name and arity.

  (f (X Y) = (cons X Y))

  (f 5) → invalid

  (f 5 6) → valid

- functions with the <u>same function name but different arities</u> are considered different functions

Marking Guideline

Marking Guide

Here is how we are going to determine partial marks.

1. [4 marks]

Your interpreter works for primitive functions. We will use the call pattern

    (fl-interp Exp nil)

to check on this. Review some examples.

2. [7 marks]

Your interpreter works for user-defined functions. This is the major part of this assignment. We will use the call pattern

    (fl-interp Exp P)

where P is nonempty to test your interpreter. We will develop a variety of test cases for this purpose. TA will provide some test cases later.

Start with a given skeleton program ...

Skeleton Program

```
(defun fl-interp (E P)
  (cond
        ((atom E) E)   %this includes the case where E is nil or a number
        (t
           (let ( (f (car E))  (arg (cdr E)) )
              (cond
                ; handle built-in functions
                ((eq f 'first)  (car (fl-interp (car arg) P)))
                .....

                ; if f is a user-defined function,
                ;    then evaluate the arguments
                ;         and apply f to the evaluated arguments
                ;             (applicative order reduction)
                .....

                ; otherwise f is undefined (not intended to be a function),
                ; the E is returned, as if it is quoted in lisp
```

Important Points to remember

- Programming Assignment is due on **Sunday, Feb 28 at 11:55 pm**
- Quiz for this Assignment is due on **Monday, March 1 at 11:55 pm**
- Submission file: **<yourID.lisp>**
- Allowed functions for programming assignment: https://eclass.srv.ualberta.ca/mod/page/view.php?id=4830015