CMPUT 325 Wi17 - NON-PROCEDURAL PROG LANGUAGES Combined LAB LEC Wi17

Assignment 2 specification

Assignment 2

Due March 2 at 11:55pm.

Submit your program as a single file named <yourID.lisp> via eClass using the **submission link at the end of this page**. Link will be active starting February 17.

In this assignment you implement an interpreter, written in Lisp, for a simple functional programming language FL, similar to the FUN language introduced at the beginning of this semester.

This assignment is worth 15 assignment marks.

Restrictions: Review the allowable built-in Lisp functions and special forms.

Note that during the development of your program, one way to avoid typing a test case repeatedly is to write a testing function and/or use setq; see examples.

Overview

A program P in FL is a list of function definitions. The FL interpreter takes such a program, together with a function application, and returns the result of evaluating the application. This evaluation is based on the principle of "replacing equals by equals". Your interpreter should be defined as a Lisp function

(fl-interp E P)

which, given a program P and an expression E, returns the result of evaluating E with respect to P.

The language FL includes a number of primitive functions that should be implemented in your interpreter. FL, as specified below, resembles a subset of Lisp without an environment -- FL does not have a special form "defun" to create permanent named functions.

Syntax of FL

In our notes, a function definition was written as

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

where f is a function, X1,...,Xn are parameters, and Exp denotes the body of the function; i.e., it defines what the function does.

For simplicity, in this assignment we will use lists to represent function definitions as well as function applications as follows:

A **function definition** is an expression

(f X1 ... Xn = Exp)

where f is the name of the function, X1 ... Xn the *parameters*, and Exp the definition of the function. The left hand side of = is called the *header* of the function and Exp the *body* of the function. For simplicity, we will use only the names X,Y,Z,M,L for parameters (variables).

Remark: the syntax for a constant function f (with no parameters) is: (f = Exp)

In this assignment, we do not consider higher-order functions.

A **program** is a list of function definitions. Review some examples.

A **function application** takes the form

(f e1 ... en)

where e1, ..., en are called *actual parameters* (or *arguments*).

For simplicity, we do not use the quote function in FL. In an application, any symbol in the place of an argument is considered a piece of data. In a list, if the first element is not a defined function, then the list is considered an input list of elements. For example, consider the following call to your interpreter:

(fl-interp '(xmember a (b c d a)) '((xmember X L = ...)))

Here, the program P defines a single function xmember for testing list membership, i.e., whether X is in L. When evaluating the expression

(xmember a (b c d a))

both a and the list (b c d a) are considered input data (as if they were quoted in Lisp). The reason is that b is not a defined function in the program P. However, if b was a function defined in P, then (b c d a) would represent the application of function b to arguments c, d and a.

A function will not be defined more than once in P. There are no free variables in FL, so a variable that appears in the body of the function must also appear as a parameter in the header of the function.

We will only test your interpreter on valid programs and expressions. For example, we will not test your program by calling an undefined function.

Primitive Functions of FL

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)
(isnumber 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.

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

Interpreter

The evaluation of an application is by "replacing equals by equals". That is, for an application (f e1 ... en), find the definition of the n-ary function f, (f X1 ... Xn = Body), in the given program, and replace (f e1 ... en) by Body with all occurrences of the parameters replaced by their corresponding arguments.

In FL, a function is identified by both its name and arity (number of arguments), and is only called with the full number of arguments. For example, if (f X Y = (cons X Y)), it cannot be used to evaluate an expression, say (f 5). (This is different from our recent lambda calculus examples where we often did that.) However, there might be another function definition with the same name but different arity, such as (f X Y Z = ...) (arity 3) or (f = ...) (arity 0).

An evaluation step described above is called a *reduction*. Reduction is performed repeatedly until no further reductions are possible. An expression that is not reducible is called a *normal form*. So, the goal of an interpreter for FL is to reduce a given expression to a normal form using the definitions in the given program.

You should implement your interpreter based on *applicative order reduction*. See orders of reductions for the interpreter to be implemented here. Review more examples.

Finally, we provide a more detailed example of reduction here.

Notes: One exception to the applicative order reduction is the IF function, for which the condition should be evaluated first, and according to the result, continue either with the THEN part or the ELSE part. Another is the Boolean functions AND and OR; always evaluate the first argument, and according to the result, either stop or continue with the second argument. E.g. with (AND E1 E2), if E1 evaluates to FALSE, stop and return FALSE without evaluating E2.

Marking Guide

Here is how we are going to determine partial marks.

1. [5 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. [10 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.

Hints: This assignment may look a bit intimidating. But it should not be the case. The main function could be coded well within a couple of pages plus some utility functions. However, you should start early, since it takes time to study the language to be implemented. You may start from this skeleton program.

Submission status

This assignment will accept submissions from **Friday, 17 February 2017, 12:00 AM**

Submission status
No attempt
Grading status
Not graded
Due date
Thursday, 2 March 2017, 11:55 PM
Time remaining
27 days 1 hour
Last modified
Friday, 3 February 2017, 10:21 PM
Submission comments
▸ Comments (0)