CMPUT 325: Non-Procedural Programming Languages

A List Interpreter in LISP

Md Solimul Chowdhury
mdsolimu@ualberta.ca

Introduction

- In assignment 2, we are implementing an interpreter for a language named FL.
  - Primitive functions.
  - User-defined functions.
- We will have a walkthrough
  - With an implementation of a **similar interpreter**
    - A restricted version of list interpreter in LISP
- Should help you in implementing your current assignment.
- Implementation of the List Interpreter has been posted
  - in the **Instructor, TA's and Labs** page in eclass)
  - https://eclass.srv.ualberta.ca/pluginfile.php/6714341/mod_page/content/89/interpreter.lisp

A List Interpreter in LISP

- The List interpreter implements 16 functions

```
; OPERATIONS
;   x                   -  a variable
;   (xquote s)          -  a constant
;   (+ e1 e2)
;   (- e1 e2)
;   (* e1 e2)
;   (/ e1 e2)
;   (rem e1 e2)
;   (atom e1)
;   (car e1)
;   (cdr e1)
;   (cons e1 e2)
;   (eq e1 e2)
;   (leq e1 e2)
 ; OTHERS
;   (if e1 e2 e3)
;   (e e1 e2 ... en)
;   (let (x1 x2 ... xk) (e1 e2 ... ek) e)
;   (letrec (x1 x2 ... xk) (e1 e2 ... ek) e)
;   (lambda (x1 x2 ... xn) e )
```

A List Interpreter in LISP

- You may categorize these functions into 2 types:
  - OPERATIONS
    - Takes at most two parameters
  - OTHERS
    - Takes more than two parameters

- Categorization helps to **reduce** the
  - Complexity of Implementations.

```
; OPERATIONS
;   x                   -  a variable
;   (xquote s)          -  a constant
;   (+ e1 e2)
;   (- e1 e2)
;   (* e1 e2)
;   (/ e1 e2)
;   (rem e1 e2)
;   (atom e1)
;   (car e1)
;   (cdr e1)
;   (cons e1 e2)
;   (eq e1 e2)
;   (leq e1 e2)
; OTHERS
;   (if e1 e2 e3)
;   (e e1 e2 ... en)
;   (let (x1 x2 ... xk) (e1 e2 ... ek) e)
;   (letrec (x1 x2 ... xk) (e1 e2 ... ek) e)
;   (lambda (x1 x2 ... xn) e )
```

A List Interpreter in LISP: Some Caveats

- Calling function to the interpreter is: **startEval**
- Accepts an expression **e** as argument

```
(startEval 'e)
```

  - quote is needed here
- has no notion of a built-in list data type.

(startEval '(car ‘(1 2))) → NIL → wrong results.

- Have to use **cons.**

(starteval '(car (cons 1 (cons 2 nil)))) → 1

Some Examples

```
* (starteval '(xquote (1 2)))

(1 2)
* (starteval '(atom x))

T
* (starteval '(car (cons 1 (cons 2 nil))))

1
* (starteval '(cdr (cons 1 (cons 2 nil))))

(2)
* (starteval '((lambda (x) (+ x 1)) 5))

6
* (starteval '(+ 5 2))

7
```

Implementation of the LIST interpreter

- **starteval** is the function that implements the interpreter.

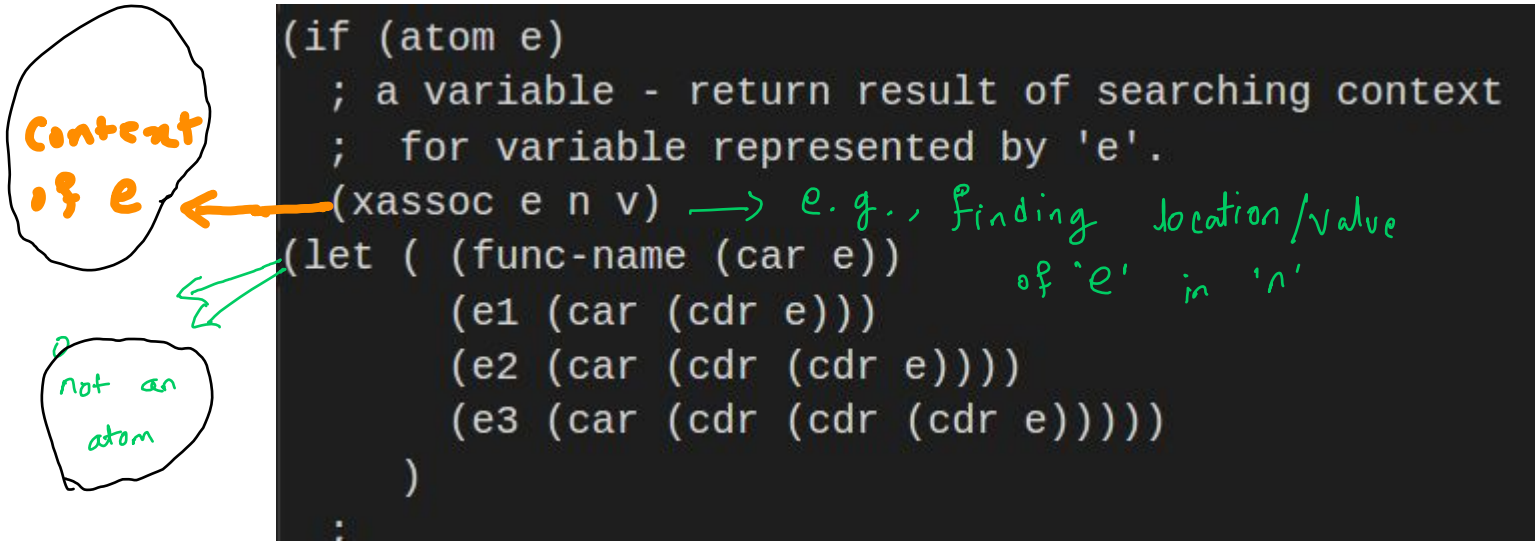
```
(defun startEval (e)
    (xeval e nil nil)
 )
```

- parameters: **e**, expression of the form shown in the previous slides
- Passes **e** to a helper function **xeval** that evaluates **e**
  - recursively, if needed
  - **xeval** is the working horse for the list interpreter
    - Comprised of a series of nested **if-else-if …** expressions to handle various cases.

Implementation of the LIST interpreter: Trivial Cases

```
(if (eq e t)
    ; t is bound to itself
  t
(if (null e)
  ; nil is bound to itself
  nil
(if (numberp e)
  ; numbers are bound to themselves
  e
```

Implementation of the LIST interpreter



Implementation of the LIST interpreter: OPERATIONS

```
    (if (eq func-name 'xquote)
        ; return the first argument unevaluated
        e1
    (if (member func-name '(+ - * / rem atom car cdr cons eq leq))
        ;
        ; Functions with 1 or 2 expressions as arguments,
;   where each argument needs to be evaluated.
        ;
        (let ( (ev-e1 (xeval e1 n v))
               (ev-e2 (xeval e2 n v))
             )
          (if (eq func-name '+)
              ; return the sum of the two evaluated arguments.
            ; obviously, the evaluated arguments should be integers.
              (+ ev-e1 ev-e2)
```

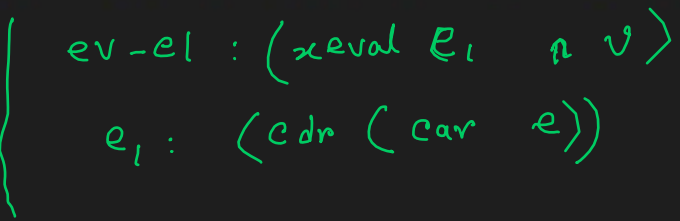
Operations

Recursive calls For Grounding e_1 and e_2

Implementation of the LISP interpreter: operations

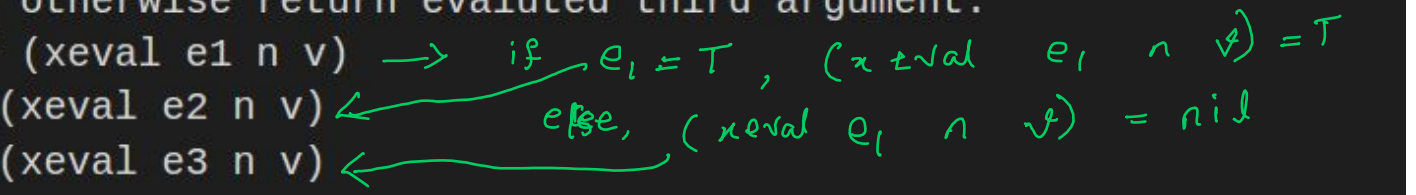
Similarly, cdr

```
)
(if (eq func-name 'cdr)
          ; return the list represented by ev-e1 with its first
          ; element removed.  If ev-e1 is an atom,
          ; return nil (not the constant nil, just nil -
          ; this will allow callers to test for execution
          ; errors)
          (if (atom ev-e1)
              nil
              (cdr ev-e1)
          )
```



Implementation of the LIST interpreter: Others

```
;
; Functions with other argument formats
;
(let ()
(if (eq func-name 'if)
          ; If first argument evaluate to true,
          ;   return evaluated second argument,
          ;   otherwise return evaluted third argument.
          (if (xeval e1 n v)
             (xeval e2 n v)
             (xeval e3 n v)
          )
```



Implementation of the LIST interpreter: Others