Zbornik radova Filozofskog fakulteta u Nišu
Serija Matematika 4(1990), 15-24

Stanimirović Predrag

EVALUATION OF LISP-EXPRESSIONS IN TURBO PASCAL

(Received 05.05.1989; Revised 05.03.1990.)

Abstract. A LISP-expression has a value if its structure corresponds to determines rules. The value of a LISP-expression is a LISP-expression. In this work we described criteria which must be satisfied by a LISP-expression if it is to have a value. We also, describe means of computing that value in a LISP-interpreter written in TURBO PASCAL.

0. Introduction

The implemented version of LISP is named MLISP. MLISP interpreter is an endless loop in TURBO PASCAL. On every pass the following actions are performed:

(1) A complete MLISP-expression is read.

(2) The entered expression is translated in an internal form.

(3) Interpreter attempts to compute its value in an internal form.

(4) If the value was successfully computed, its internal form is translated to the external form.

The interpreter is made up from the function **eval** and other functions which implement the MLISP-functions. The main part of the interpreter is **eval**. Its argument is the internal form of an expression, and its result is the internal form of of the value of the expression.

Process of taking a symbolic expression, or list, and performing a computation on it is called evaluation. The result of evaluation of a MLISP-expression is named its value.

AMS Subject Classification (1980): 68F99

1. The internal form of MLISP-expressions

Cells are the most important LISP-objects. A cell is a compound of two objects. The first object is called car (head) and the second is called cdr (tail). In PASCAL a cell is represented by a pointer variable which is defined as a record containing two pointers.

The pointer variabble **x^** which represents a cell contains the following two fields:

- **x^.left** is a pointer which points to the *car*;
- **x^.right** is a pointer which points to the *cdr*.

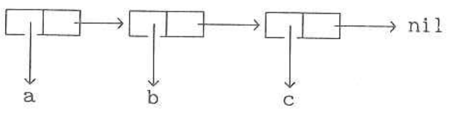
1.1. The internal form of lists

A list is a sequence of MLISP-expressions written in a pair of parentheses. Elements of a list are separated by spaces.

List are represented using binary trees. Binary tree have nodes with exactly two pointers in them, i.e. have cells. One of the pointers points to the left subtree, and one points to the right subtree.

Binary trees have terminal nodes. In the case of LISP, terminal nodes are atoms. LISP uses binary trees to represent lists as follows: The left subtree of a node points to the first element (*car*) of a list, and the right subtree points to the rest (*cdr*) of list.

EXAMPLE: Internal form of the list (a b c) is represented as the following binary tree:



The direction of the pointers is from the left to the right and downward.

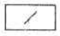
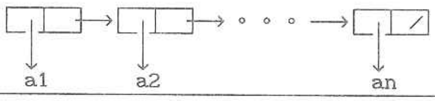
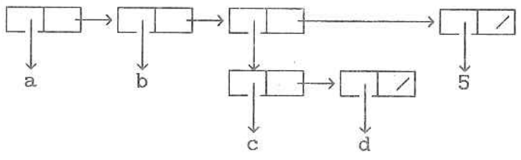
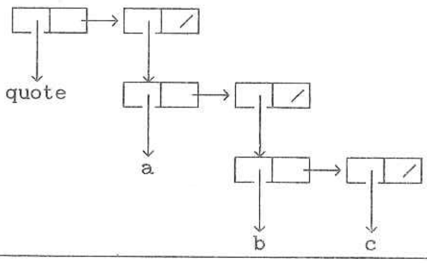
Lists can be nested. If an element of a list is itself a list, the left pointer of corresponding node to the top level list points to the binary tree which itself represent a list.

If the binary tree identified by the pointer **x** represented the internal form of a list, then:

(1) **x^.left** is the left branch of the binary tree and represents the internal form of the head (first element) of the list.

(2) **x^.right** is the right branch of the binary tree and represents the internal form of the tail (rest) of the list.

EXAMPLE: Graphic illustration of internal forms of some lists.

lists	illustration of the internal forms
nil	
(a1 a2 ... an)	
(a b (c d) 5)	
(quote(a (b c)))	

1.2. Internal form of atoms

An atom is a LISP object which is not a cell.

The internal form of an atom is a pointer variable which is defined as a record. The internal form **x^** of an atom is a record with the following field list:

- **x^.left**;
- **x^.right**;
- **x^.type**;
- variant part.

For all the types of atoms is **x^.left = nil** and **x^.right = nil**. This indicate that the pointer variable **x^** is the internal form of an atom.

In this way, an atom is represented as a cell whose *car* and *cdr* are equal to nil.

The field **x^.type** is defined as follows:

(1) is **x^.type** = **strin** for strings;
(2) is **x^.type** = **name** for built-in functions;
(3) **x^.type** = **num** for integers;
(4) **x^.type** = **sym** for variables, user-defined functions and symbols.

The variant part of the record **x^** is defined as follows:

(1) If **x^.type** = **strin** or **x^.type** = **name** or **x^.type** = **sym** there is a field **x^.a**.

(2) If **x^.type** = **num** there is a field **x^.r**.

The field **x^.r** is an integer, and field **x^.a**, is a string containing the name of the atom with the internal form **x**.

2. Evaluation of lists

In the programming language MLISP the head of the list is an operator, and the tail represent list of the arguments. The list evaluates applying the operator to the arguments.

An operator can be :

(1) *subr*;

(2) *fsubr*;

(3) a user-defined function.

The operator is a *subr* if it is applied to the evaluated arguments; the operator is a *fsubr* if it is applied to the unevaluated arguments.

2.1. An operator which is *SUBR*

The remaining list of elements are evaluated and passed as arguments to the corresponding function.

Internal forms of values of the arguments are formed in the procedure **evlis**. During evaluation of an argument the function **eval** is called.

Any argument is or an atom or a list and has an operator and arguments.

Internal forms of the opertors are stored in the array **oper**. Internal forms of values of arguments of an operator are stored in the corresponding element of the array **argument**.

Variable **d** is used as an index into these arrays. The value of **d** is incremented by one when the left parenthesis is detected, and decremented by one when the right parenthesis is detected. The initial value of **d** is one.

EXAMPLE. For the initial expression (+ (- 2 (* 3 4 1)) (* 3 5)) the value fo **d** is 1.
For subexpressions (- 2 (* 3 4 1)) and (* 3 5) the value of **d** is 2.
Finally, for (* 3 4 1) the value of **d** is 3.

The operator corresponding to **d** is **oper[d]^.a**. The corresponding functions in TURBO PASCAL are called using elements of the queue **argument[d]** as arguments.

2.2. Procedure EVLIS

Input to this procedure is the internal form of the tail of the the evaluated list. The outputs are:

- the queue **argument[d]** whose elements are the internal forms of values of the subexpressions corresponding to **d**, and

- the number of evaluated arguments corresponding to the **d**, denoted as **brv[d]**. The value of variable **brv[d]** is used in the syntax analysis.

While the queue **argument[d]** is formed, pointers **arg[d]** and **arg1[d]** are used. The pointer **arg[d]** points to the first element of the queue **argument[d]**, while **arg1[d]** points to the last element of the queue.

A description of the procedure **evlis** with formal parameter **x** follows:

STEP 1. Set the initial values:

*__brv[d]__ :=0;

*__arg[d]^.left:=nil; arg[d]^.right:=nil;__

*__arg1[d]:=arg[d]__.

STEP 2. A while-loop which terminates in either of two cases:

*all of arguments are evaluated, or

*a syntax error was detected in argument, i.e. the argument evaluates to the pointer **err**.

In the body of the while-loop STEP A and STEP B are executed.

STEP A.

If then the current argument is an atom then A1 is executed, else A2 is executed.

A1. Evaluate the atom using **eval**.

A2. (1) Increment **d** by one.

(2) Evaluate the list.

(3) Decrement **d** by one.

STEP B.

If the internal form of the value of the current argument is not equal to **err**, the following steps are done:

2*

(1) Abandon the evaluated argument.
(command **x:=x^.drugi**).

(2) Place the internal form of the value of the current argument at the end of the queue **arg1[d]**:

(3) Increment **brv[d]** by one.

STEP 3. If the while-loop is terminated after **x=nil** set **argument[d]:=arg[d]^.drugi**.

2.3. An operator which is *FSUBR*

In such a case two methods of evaluation are used, denoted as A1 and A2:

A1. Corresponding function in TURBO PASCAL is caled with the internal form of the tail of the starting list used as argument.

EXAMPLE: Expressions with the function **cond** as a head are evaluated in this way.

A2. The internal form of value is taken as equal to the corresponding pointer from the internal form of the tail of the list.

EXAMPLE: If the head of the expression is the function **quote**, and the internal form of the tail is the pointer **a2**, then the internal form of the value is equal the pointer **a2^.prvi**.

2.4. Definition and use of user-defined functions

That is once a function defined, we can put it in the begining of a list, and the interpreter will apply the function to those arguments.

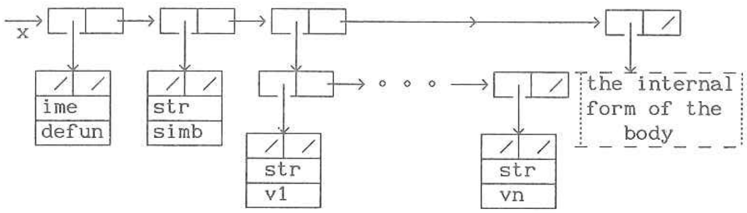
A function is defined by an expression with the function **defun** as a head. A call to **defun** look like this:

(defun symbol (v1 v2 ... vn) (body). (2.1.)

Such an expression is called the functional definition of a symbol. Elements of this expression have the following meaning:

1. The atom **symbol** is the name of the user-defined function.
2. (v1 v2 ... vn) is list of formal parameters.
3. **body** is a MLISP-expression. It is caled the body of the functional definition and will be used in aplication of the function.

The internal form of this expression is illusstrated here:



The result of evaluation of this expression is the atom **symbol**.
We can call **symbol** as follows:

(symbol arg1 arg2 ... argn). (2.2.)

Every user-defined function gets an unique number j (starting from 2) associated with it. This is achiewed by the following actions:

A1. When the interpreter is started, **j** is inicialized to one.

A2. During evaluation of expression (2.1.) **j** is incremented by one. Variable **j** is used:

*-as an index into array **names**, containing names of user-defined functions;

*-as an index into array **v**, containing internal forms of bodies of functional definitions.

During the evaluation of expression (2.1.) the interpreter executes steps A, B, C, D and E as follows:

A. The value of **j** increment to one.

B. Name **symbol** is assigned to the variable **names[j]**.

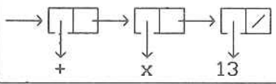
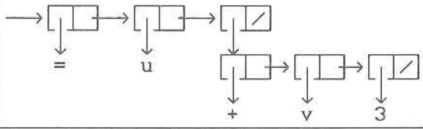
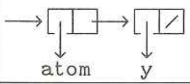
C. Variable **nvar[j]** contains the number of variables in the last functional definition. At this point it is initialized to zero.
For each variable **v1** to **vn** the following steps are performed:
*-**nvar[j]** is incremented by one.
*-Name of variable is assigned to **var[j,brp[j]]**.

D. The internal form of the expression **body**, i.e. **x^.right^.right^.right^.left** is assigned to the variable **v[j]**.

E. The internal form of the value of the expression (2.1.) is equal to the pointer **x^.right^.left**.

EXAMPLE: Graphic illustration of values of the variable j and arrays **names** and **var** after evaluation of the following expressions:

```
>(defun s(x) (+ x 13))
>(defun jed(u,v) (= u (+ v 3)))
>(defun s(y) (atom y))
```

j	imena	v	prom
2	s		prom[2,1]=x
3	jed		prom[3,1]=u prom[3,2]=v
4	s		prom[4,1]=y

During the evaluation of the expression (2.2.) the interpreter executes steps A1 and B1, as follows:

A1. Atom **symbol** is compared with the elements of the array **names** and index **l** is decremented by one. The initial value of l is j. Search is terminated when **names[l]=symbol**. The value of l, found in this way, corresponds to the last functional definition of the symbol **simb**. The internal form of the value of the expression (2.2.) is equal to **eval(v[l])**.

B1. Evaluation of the formal parameter **vi** during evaluation of the expression **v[l]** is performed in the following way:

STEP 1.
Evaluate each of the arguments **arg1** to **argn** using the procedure **evlis**.

STEP 2.
Compare atom **vi** with the elements of the aray **var[l,l2]** and decrement index **l2** by one. Initial value of **l2** is **nvar[l]**. Search is terminated when **var[l,l2] = vi**.

STEP 3.
The internal form of the value of the formal parameter **vi** is equal to the corresponding value in the queue **vr**:

3. Evaluation of atoms

Integers, strings, subrs, and fsubrs evaluate to themselves. The internal forms of values of these atoms are equal to the internal forms of these atoms. In practice, if the pointer **x** is the internal form of such an atom, then **eval(x)=x**.

The value of a symbol in MLISP is the value of expression which bound to that atom. MLISP has the special function **set** which binds the value of one of one's own arguments to the other argument. For example, the equvalent of "r ⇐ (a (b) c)" is accomplished in MLISP by:

```
>(set (quote r) (quote(a (b) c)))
@ ( ( b ) c )
```

We can query MLISP about value of **r**:

```
>r
@ (a ( b ) c )
```

The first arguments in the expressions which head is the function **set** are elements of the array **stack**. The internal forms of the values of the second arguments are elements of array **v1**.

The elements of arrays **stacj** and **v1** are defined paralelled. An element of array **stack** and the corresponding element of array **v1** logically constitute an ordered pair.

Evaluation of the symbol **r** proceeds as follows.

Compare atom **r** with the elements of the array **stack** and decrement index **l1** by one. Search starts with l1:=**j1**. Search is terminated when **stack[l1]** = **r**. The internal form of the value of the the atom **r** is equal to **v1[l1]**.

The arrays **v1** and **stek** are generated by the following algoritm:

STEP 1.

During the evaluation of the list with the function **set** as a head, variable **j1** is incremented by one.
Initial value of **j1** is one. The value of **j1** is an index into arrays **stack** and **v1**.

STEP 2.

The first argument of the function **set** is assigned to the variable **stack[j1]**. The internal form of the value of the second argument is assigned to the pointer **v1[j1]**.

REFERENCES

[1] LAVROV & SILAGADZE, *Jazik LISP i ego realizacija*, Nauka, Moskva, 1981.

[2] J. FODERARO, *The Franz LISP manual*, University of California, Berkely California, 1979.

[3] DAVID GRIES, *Compiler Construction for Digital computers*, Cornell University, Toronto 1971.

[4] ROBIN HUNTER, *The design and construction of compilers*, New York, 1983.

[5] P. STANIMIROVIĆ, *Implementacija LISP interpretatora u TURBO PASCALu,* magistarski rad.

[6] TURBO PASCAL 4,0, USERS GUIDE, Osborn, McGraw Hill, New York.

[7] ROBERT WILENSKY, Common LISPcarft, University of California, Berkeley, W.W.Norton & Company, New York, London.

P. Stanimirović

EVALUACIJA LISP-IZRAZA U TURBO PASCALU

U radu su izložene teorijske osnove evaluacije LISP izraza. Takođe, opisana je praktična evaluacija LISP- izraza u TURBO PASCALu.

Filozofski fakultet
Ćirila i Metodija 2
18000 Niš
Yugoslavia