Zbornik radova Filozofskog fakulteta u Nišu
Serija Matematika 4(1990), 71-78

Stanimirović Predrag

LEXICAL ANALYSIS OF LISP-EXPRESSIONS IN TURBO PASCAL

(Received 17.03.1990.)

Abstract. In the interpreter of programming language MLISP, the process of lexical analysis executes before of syntax analysis and evaluation.

The process of lexical analysis in this interpreter contains the following stages:

- determination the length of an atom;
- detection of the atom;
- building a specific internal form of the atom.

0. Introduction

Any inetrpreter or compiler executes lexical analysis of the input program. A lexical analysis can be independent stage of interpretation or compilation, or it is called dyring the syntax analysis and the runing of a program.

During the lexical analysis, the source program is accepted as a character string and lexical units contained in the language statements are detected. The lexical units are called tokens or atoms.

Lexical analyzer usualy detects keywords, operators, identifiers, integrs and real numbers, strings and other lexical units of the input program. The list of atoms which is detected depends on the programming language implementad by the interpreter or compiler.

The output of a lexical analysis is a series of atoms. Because of the increase of effectiveness of the next stages of interpretation or compilation, each atom is represented by a corresponding internal form.

AMS Subject Classification (1980): 68F99

1. The lexical analyser of the programming language MLISP

The lexical analyser of the programming language MLISP detects the atoms of that language and transforms them into a specific internal form.

MLISP lexical analyser does not analyse complete source program in single call. It also contains two functions which are called during the process of transformation of the entered MLISP-expression into the internal form. The result of calling the lexical analyser will be the internal form of the next atom of the entered expression.

2. The grammar for the lexical analysis

The grammar for the lexical analysis defines the atoms detected in the lexical analyser. BNF notation is used for the description of the grammar.

2.1. Constant with fixed values

Two constants with fixed the fixed value are used.

A. **nil** is the most important simple LISP data type.

In MLISP, **nil** is used in the following ways:

- **nil** is the usual list terminator (or, more precisely, **nil** represents an empty list).
- **nil** represent the logical constant **false**. Any non-**nil** value being often thought of as **true**.
- **nil** is also used as the default result value by some special function such as **cond**.

B. The constant **t** represent value of non-**nil** expression.

<special__constants> ::= nil|t

2.2. Strings

A string is specified as a sequence of characters enclosed in double quotes.

<string> ::= "<array_chars>"

<array_chars> ::= ε|<char><array_chars>

<char> ::= <letter>|<digit>|<special_char>

<letter> ::= a|b|...|z

<digit> ::= 0|1|2|3|4|5|6|7|8|9

<special_chars> ::= ~|!|@|#|$|%|*|(|)|+|=|-|?|

2.3. Left and right parentheses

The left and right parentheses represent individual atoms and separators.

<parentheses> ::= (|)

2.4. Integers

An integer is represnted by a sequence of characters starting with either from plus or minus sign, or a digit 0..9, and containing only digits.

<integer> ::= [+|-]<integer_out>
<integer_out> ::= <digit>|<array_digits>
<array_digits> ::= ε|<digit><array_digits>

2.5. Build-in functions

Built-in functions represent the atoms whose detection make that determined function in TURBO PASCAL are called.

<name_of_function> ::= +|-|*|/|=|>|<|eq|<=|>|/=|or|if|1+|1-|
car|cdr|not|and|set|abs|neq|cons|cond|
atom|null|caar|cadr|cdar|cddr|list|eval
last|exit|defun|quote|zerop|plusp|print|
prog1|prog2|progn|listp|minus|ncons|
minusp|numberp|stringp|symbolp|length|concat
|substring

2.6. Symbols

A symbol is represents by its name.

<symbol> ::= <array_chars>

3. The internal form of atoms

The internal form of an atom is a pointer variable representing a record. The fields of the record are defined so that they carry complete information detected by the lexical analyser.

An exception are the constants with a fixed value. The internal form of the **nil** is the pointer **nil**, while the internal form of the constant **t** is the reserved pointer **t**.

Let the pointer variable x^ represent the internal form of an atom. The two fields of the record x^ are defined in the same way for all the types of atoms:

x^.left:=nil; x^.right:=nil.

The field **x^.type** takes the values of the enumerated type **typesizraz = (name, num, strin, sym)**. The value of the field **x^.type** determined the type of the atom.

A. The internal form of a string contains the following fields:

- **x^.left := nil;**
- **x^.right := nil;**
- **x^.type := strin;**
- the field **x^.a** represents the concatenation of the caracters forming the string.

B. The fields of the pointer variable **x^**, which represents the internal form of an integer, are defined as follows:

- **x^.left := nil;**
- **x^.right := nil;**
- **x^.type := num;**
- the field **x^.r** represent the numeric value of the atom.

C. If the pointer variable **x^** represent the internal form of a built-in function, then:

- **x^.left := nil;**
- **x^.right := nil;**
- **x^.type := name;**
- the field **x^.a** is defined as the concatenation of the characters forming the atom.

D. If **x^** represents the internal form of a symbol, its field are defined in the following way:

- **x^.left := nil;**
- **x^.right := nil;**
- **x^.type := sym;**
- the field **x^.a** represents the concatenations of the characters forming the symbol.

4. The idea of lexical analysis in MLISP

Using of ponters and pointer variables give the following advantage:

- Lists are basic structure in LISP, and pointers are the best for their representation.
- Pointers can be composed in binary trees.
- Binary trees are adequate for the process of syntax analysis and the evaluation of LISP-exressions. The car represented the internal form of a function, and cdr contains the internal forms of arguments.
- A pointer variable stored on the heap can be disposed.

Before calling the lexical analyser, a complete MLISP-expression is entered and put into the array **a**. The **a** is used as the input buffer. Elements of the **a** is characters: each element of the **a** corresponds to the character of the expression. The elements of the array **a** are the input values of the lexical analyser.

The lexical analyser in the MLISP interpreter consists of the functions **reada** and **read1**.

At the first call, the functions **reada** and **read1** use the array **a** starting from its first element, and, at each next call - from the place where the previous analysis has stopped.

Characters from the buffer **a** are extracted in the function **reada**. This function defines the number of the buffer-elements forming an atom, i.e. his length.

The function **read1** use the length of an atom and forms his internal form.

4.1. The function READA

The function **reada** "separates" a "portion" as big as an atom from the **a** and copmputes its length.

The meanining of the variables in the function:

- The value of the variable **m** represents the current length of the separated atom.
- The value of the variable **k** represents the index of the buffer **a**.

Algorithm is described as follows:

STEP 1.

The starting value of the variable **m** is zero.

STEP 2.

Increase the value of the variable **k** by one.

STEP 3.

All the spaces and CR and LF sequences is ignored starting from the **k**.

STEP 4.

CASE 1. If **a[k]** = '(' or **a[k]** = ')' the result of the functions is one.

CASE 2. If **a[k]** = '"' then is defined a repeat-until loop which terminates when **a[k]** = '"' or **a[k]** = ' ' or **a[k]** = '(' or **a[k]** = ')'.

In the body of the loop increase the values of the variables **k** and **m** by one.

If **a[k]** = '"', the value of the variable **m** represent the length of the string, else it represents the length of the symbol.

CASE 3. For other values of **a[k]** the steps A and B are executed C.

STEP A.

A while-loop which finites if **a[k]** is equal to an separator, i.e. **a[k]** = ' ' or **a[k]** = '(' or **a[k]** = ')' or **a[k]** = #13.

In the body of the loop increase the values of the variables **k** and **m** by one.

When is the while-loop terminated, the value of the variable **m** represents the length of the atom.

STEP B.

Decrease the value of **k** by one in order to make the lexical analysis start from aseparator at the next call.

4.2. The function READ1

The function **read1** detects an atom and builds the corresponding internal form. The pointer which point to the internal form of the detected atom is assigned to the name of the function. The length of the atom, assigned to the name **reada**, is used during his detection.

For the constant **nil** set up **read1 := nil**, and for the constant **t** set up **read1 := t**. In this way, the internal form of these atoms is assigned to the name of the function.

For other atoms the pointer **p** is defined. This pointer points to the internal form **p^** of the atom.

The dynamic allocation procedure **new(p)** alocates a new memory area in the pointer variable **p^**. The pointer **p** points to the pointer variable **p^** to the specific memory adress.

Two fields of the record **p^** are the same for the all the types of atoms:

p^.left:=nil; p^.right:=nil.

The remainding fields of that record are defined after the atom detection.

Command **read1:=p** the name of the function **read1** set up to the internal form of the detected atom.

A. THE DETECTION OF THE ATOMS NIL AND T

The atom **nil** is detected by using **l=reada=3** and **a[k-2]+a[k-1]+a[k]='nil'**.

For the detection of the constant **t** is used **l=reada=1** and **a[k]='t'**.

B. THE DETECTION OF STRINGS

An atom of the length l represents a string if **a[k-l+1]='"'** and **a[k]='"'**. The remaining fields of the dynamic variable **p^** are defined in the following way:

p^.type:=strin;

C. THE DETECTION OF BUILT-IN FUNCTIONS

An atom of the length l represents a built-in function if the string a[k-l+1]+ ∘ ∘ ∘ +a[k] is equal to one of the built-in functions in the implemented language.

If the atom represents a built-in function, the remaining fields of the ponter variable **p^** are defined as follows:

p^.type:=name;

p^.a:=a[k-l+1]+ ∘ ∘ ∘ +a[k].

D. THE DETECTION OF INTEGERS AND REAL NUMBERS

A real number can be detected using the procedure **val(s,re,code)** in TURBO PASCAL. This procedure converts the string value

s=a[k-1+1]+ ∘ ∘ ∘ +a[k] to its numeric representation and stored the result in re. If the string s is somehow invalid, the index of the offending character is stored in code; otherwise, code is set to zero.

A number re represent an integer if trunc(re)=re.

Remaining fields of the internal form of such an atom is defined in this way:

```
p^.type:=num;
p^.r:=re.
```

F THE DETECTION OF SYMBOLS

The other atoms are symbols.

Remaining fields of the internal form of an symbol are defined as follows:

```
p^.type:=sym;
p^.s := a[k-1+1] + ... + a[k].
```

REFERENCES

[1] LELAND L. BECK, *An introduction to systems programing*, Addison-Wesley Publishing Company, San Diego State University, 1987.

[2] ROBERT WILENSKY, *LISPcraft*, WWNORTON & company, New York, London, 1984.

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

[4] ROOBIN HUNTER, *The design and constructions of compilers*, New York, 1983.

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

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

Predrag Stanimirović

LEKSIČKA ANALIZA LISP-IZRAZA U TURBO PASCALU

U radu su izloženi osnovni principi i uloga leksičke analize u konstrukciji interpretatora ili kompilatora. U tom kontekstu date su osnovne koncepcije i specifičnosti leksičke analize jedne varijante programskog jezika LISP u PASCALu.

Filozofski Fakultet
Ćirila i metodija 2
18000 Niš
Yugoslavia