CS 92

MLISP

BY

HORACE ENEA

TECHNICAL REPORT NO. CS 92
MARCH 14, 1968

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

MLISP[1]

by

Horace Enea

March 14, 1968

Computer Science Department

School of Humanities and Sciences

Stanford University

[1] This research was supported by Grant MH 06645-06 from the National Institute of Mental Health.

Introduction

Mlisp is an Algol-like list processing language based on Lisp 1.5 . It is currently implemented on the IBM 360/67 at the Stanford Computation Center, and is being implemented on the DEC PDP-6 at the Stanford Artificial Intelligence Project. The translator produces an object program in Lisp 1.5 S-expressions. at a speed of 1800/2000 lines of Mlisp per minute.

The principle reason for writing Mlisp was to provide a good list processing language with a convenient notation, a higher degree of machine independence, and string facilities. The balance of this paper will be a very informal presentation of the language so that the reader will be able to run programs in Mlisp with a minimum of effort. The language has an extremely simple syntax which is presented in Appendix I. The style of presentation will be by example. It is assumed the reader knows Lisp 1.5 and is familiar with Algol. All the functions of the underlying Lisp processor are available to the user, and therefore, the user should consult the Lisp/360 manual in addition to this presentation. Additionally, the storage conventions are, of course, those of the underlying processor; that is, limitations on the length of printnames of atoms, conventions for numbers, handling of bindings, etc. These points are generally not important to the Mlisp user since the translator knows these conventions and produces appropriate S-expressions.

The Sample Program

```
Line No.     Text

   1             % THIS PROGRAM IS AN EXAMPLE %
   2
   3         BEGIN
   4         NEW A,B,C,D,I,J,L,ST;
   5         MACRO OFF "VERBOS (NIL)";
   6         C:=(6.7*4)/3 + 2**4;
   7         A:=C CONS D;
   8         B:=<1,2,3>;
   9         B:=C:=CDR(A);
  10         B:='B;
  11         C:='( B (C.A));
  12         PRINT (ST:="AB//  7"@DBQUOTE);
  13         PRINTSTR(ST SUBSTR <3,2>);
  14         C:=MAKEATOM("ASDF//");
  15
  16         LAST:=#L:
  17            IF¬L THEN NIL
  18            ELSE IF ¬ CDR(L) THEN CAR(L)
  19            ELSE LAST(CDR(L));
  20
  21         REVERSE:=#L:
  22            BEGIN NEW I,J;
  23-           FOR I IN L DO
  24                IF ATOM(I) THEN J:=I ¢ J
  25                ELSE J:= REVERSE(I) ¢ J;
  26            RETURN( J) ;
  27         END;
  28
```

Line No.	Text
29	RR:=#:<READ(),READ()>;
30	
31	A:=DO I :=I+1 UNTIL FN(I);
32	B:=COLLECT <I:=FN(I)>UNTIL I EQ 'END;
33	WHILE ¬((A:=READ()) EQ 'END) DO INPUT(A);
34	C:=WHILE ¬((A:=READ()) EQ 'END) COLLECT <A>;
35	FOR I ON L DO FN(I);
36	J:=FOR I IN L DO FN(I) UNTIL QN(I);
37	FOR I IN 1 BY 4 TO 13 DO FN(I);
38	FOR I IN 1 TO 10 DO FN(I);
39	J:=FOR I IN L COLLECT FN(I);
40	
41	J:=FN(FUNCTION(+), FUNCTION(TIMES));
42	
43	J:=<<3,2>,<4,<6,8>>> SUB <2,1>;
44	J:=F7(<1,2,3,4,5,6,7,8,9,0>);
45	OFF;
46	END.
47	(Input follows end.)

Explanation of Sample

Lines 1 to 47 represent a collection of all the features of Mlisp. The program does not compute anything.

Line No. — Text

1 Comments --

Anything except a "%" between two "%"s is a comment and is ignored by the translator.

Blanks --

Blanks may be used between any identifiers or special symbols to improve readability.

Programs --

Mlisp programs generally start with a BEGIN and finish with an END (See Appendix I for a definitive specification of the syntax.). Each BEGIN-END pair constitutes a program. The value of a program is NIL unless there is a RETURN within the BEGIN-END pair. (See line 26 for an example.) All BEGIN-END pairs are translated into Lisp programs, and therefore, return a value even if they have no local variables.

4 NEW --

Program variables are bound by the NEW declaration. Their initial value is NIL.

5 MACROs --

A simple substitution macro facility is provided so that line 45 will be expanded into:

```
VERBOS(NIL);
```

Line No. Text

5 and then translated. If another macro call is discovered it is expanded, therefore, recursive expansion is possible. Macros must be declared before they are used.

6 Arithmetic --

Line 6 translates into:

```
    (SETQ C (QUOTIENT (TIMES 6.74)
          (PLUS 3 (EXPT 2 4 ))))
```

Notice that hierarchies are right to left. Unary operators (- + ¬) are translated before binary operators. Any operator that is not unary is binary.

7 Infix operators --

Any function or operator which takes actly two arguments may be used as an infix operator. The translator recognizes the following abbreviations:

Mlisp	Lisp
¢	C ONS
@	APPEND
=	EQUAL
*	TIMES
	DIFFERENCE (or MINUS when used in unary position)
&	AND
+	PLUS
/	QUOTIENT
|	OR

Line No. Text

7

Mlisp	Lisp
ı	NOT (this is equivalent to NULL)
**	EXPT
:=	SETQ (when used as operator)

8 Lists --

a, 2, 3> translates into:

(LIST 1 2 3)

a, <2, 3>> translates into:

(LIST 1 (LIST 2 3))

9 Multiple assignment --

Multiple assignment statements are allowed.

10-11 Quoting S-expressions --

S-expressions are preceded by a single quote mark (') and follow the syntax of Lisp except that special characters may not be used within an atom. (See line 14 for a description of how to create atoms containing special characters.)

12 Strings --

"ABC" is an abbreviation for '(A B C) ; however, special characters may appear between quotes ("), and will be handled correctly. DBQUOTE has as its value:

'(")

therefore, by using the APPEND operator (@) any string may be created. Line 12 would produce

Line No.   Text

12

```
(A B / /    7 " )
```

as output. By using the intrinsic function PRINSTR instead of PRINT we would have gotten:

```
AB//  7"
```

13 Substrings --

The intrinsic function SUBSTR takes two arguments; a string, and a list of two integers (starting position and number of characters to be extracted). The value of

```
"ABCDEF" SUBSTR <3,2>
```

is "CD". PRINT will print this as:

```
(C D)
```

and PRINTSTR will print this as:

```
CD
```

14 MAKEATOM and STR --

MAKEATOM takes a string as its argument and produces an atom with that string as its printname. (See line 12). STR takes an atom or a number and makes a string of its printname.

16- 19 Defining functions --

On line 16 LAST is defined to be a function. The name of the function being defined need not be declared NEW. Sharp (#) stands for LAMBDA. A function with three arguments would start:

```
FN:=#A,B,C:
```

Line No. Text

16-19 The translation of lines 16-19 is:

```
(DEFINE(QUOTE ((LAST
     (LAMBDA (L)
           (COND
                ((NOT L) NIL)
                ((NOT (CDR L)) (cm L))
     )     )
))))
```

Remember in Lisp NOT and NULL are the same.

21-27 Another function is defined --

This function reverses a list. J is the value of the function. I becomes successively:

```
CAR(L), CADR(L),...
```

until L is exhausted. When L is exhausted, the FOR expression (See 37-39) terminates and the value of I becomes NIL outside the FOR expression. Notice that a RETURN is needed for each BEGIN-END pair; for example:

```
A:=BEGIN
     RETURN(BEGIN
         ...
         RETURN(L);
         END);
   END;
```

Line No. Text

29 A function with no argument is defined --

RR, a function with no arguments is defined.

READ, a function with no arguments is called.

31 DO-UNTIL --

Every expression has a value including control expressions.

FORM:

```
      DO expression1 UNTIL expression2
```

EVALUATION:

```
   A) V ← value (expression1).
      B ← value (expression2).
      if B ≠ NIL then return (V).
      go to A.
```

32 COLLECT- UNTIL --

FORM:

```
      COLLECT expression1 UNTIL expression2
```

EVALUATION:

```
      V ← NIL.
   A) V ← V append value (expression1).
      B ← value (expression2).
      if B ≠ NIL then return (V).
      go to A.
```

Notice that the value of expression1 should be a list, and that expression1 may be considered as an "example" of the value of the COLLECT-UNTIL.

Line No. Text

33 WHILE-DO --

WHILE-DO is evaluated similarly to DO-UNTIL except that the test is performed first.

FORM:

WHILE expression1 DO expression2

EVALUATION:

```
      V ← NIL.
  A) if value (expression1) = NIL then
           return (V).
      V ← value (expression2).
      go to A.
```

34 WHILE-COLLECT --

WHILE-COLLECT is like COLLECT-UNTIL except the test is performed first.

FORM:

WHILE expression1 COLLECT expression2

EVALUATION:

```
      V ← NIL.
  A)  if value (expression1) = NIL then
           return (V) .
      V ← V append value (expression2).
      go to A.
```

35 ON --

ON may be substituted for IN in any FOR expression. When ON is used 'I' becomes successively L, CDR(L), CDR of that, etc., until L is exhausted.

Line No. Text

36 UNTIL in FOR expression --

By adding an UNTIL clause to a FOR expression an additional test may be performed. The FOR expression will terminate if the value of the UNTIL becomes true (non-NIL). The value of 'I' is the last value assigned to it.

37-38 STEP-FOR --

In line 37 'I' becomes successively 1, 5, 9, 13. In line 38 the BY is omitted and is, therefore, understood to be 1.

39 FOR-COLLECT --

COLLECT may be substituted for DO in FOR expressions with the same effect as in WHILE-COLLECT or COLLECT-UNTIL. Notice that 'FN(I)' must return a list as its value.

41 Functional arguments --

Functional arguments are passed via the pseudo-function FUNCTION as in Lisp 1.5.

43 A subscription function --

A subscripting function called SUB is available. The value of 'J' in line 43 is 4. The first argument is the list to be subscripted and the second is a list of subscripts. An 'out of range' subscript returns NIL. Subscripts may not be used on the left of a ':='.

Line No. Text

44 Fields --

A set of functions F1, F2,...F9 is available corresponding to CAR, CADR, . . .CADDDDDDDDR respectively. The values of 'J' in line 44 is 7. A field function may not be used on the left of a ':='.

46 Ending a program --

After the final END in a program put a period (.) .

47 Data --

Input data directly follows the END card. No special cards are needed after the last data card.

Appendix I

The syntax of Mlisp

The syntax is in Backus-Naur form with the following additions:

[1 enclosed construction is optional

{ 3 alternative possibilities enclosed

... preceding may be repeated any number of times

```
<prog>     ::=  <expr>

<expr>     ::=  <simpex> [<opr> <expr>]...

           ::=  <empty>

<simpex>   ::=  BEGIN [<decl>;]...[<expr>[;
                     <expr>]...]END

           ::=  IF <expr> THEN <expr> [ELSE <expr>]

           ::=  FOR <id> {IN} <expr> <fortail>
                         {ON}

           ::=  {DO     }  <expr> UNTIL <expr>
                {COLLECT}

           ::=  WHILE <expr> {DO     }  <expr>
                             {COLLECT}

           ::=  <[<expr> [, <expr>]...]>

           ::=  "[<any character not">]..."

           ::=  (<expr>)

                  +
           ::=              <simpex>
                i ⌐ 1

           ::=  '  <sexp>

           ::=  <id>  [<exprl>]

           ::=  <number>

<opr>      ::=  | * | - | / | + | ¢ | | | & | @ | **

           ::= # [<id> [, <id>]...]: <simpex>

           ::=  <id>

<decl>     ::=  NEW <id> [,<id>]...

           ::=  MACRO <id> "[<token>]..."
```

```
                           {DO     }
<fortail>  ::=  [<by opt>] {       } <expr> [UNTIL <expr>]
                           {COLLECT}

<by opt>   ::=  [BY <expr>] TO <expr>

<token>    ::=  <id>

           ::=  <number>

           ::=  <any special character except " >

<exprl>    ::=  (<expr2> [,<expr2>]...)

           ::=  [#[<ID>]] : ] <expr>

<expr2>    ::=  FUNCTION (<opr>)

           ::=  <expr>

<sexp>     ::=  (<sexp2>

           ::=  <token>

<sexp>     ::=  )

           ::=  <sexp> <sexp3>

<sexp3>    ::=  .   <sexp> )

           ::=  <sexp4>

<sexp4>    ::=  )

           ::=  <sexp> <sexp4>
```

Appendix II

Initial Conditions

The garbage collector printout is turned off; it may be turned on by saying:

```
VERBOS(T);
```

Job Setup

```
          Wylbur --                                         Col. 72

//XXXX JOB (nnnn, bin, r, ℓ),'your name',MSGLEVEL=1
//JOBLIB DO DSNAME=SYS2.PROGLIB,DISP=OLD
//JOBLIB EXEC PGM=LISPA
//LISPOUT DD SYSOUT=,DSNAME=nnnn.pppp,UNIT=2314                 C
//                VOLUNE=SER=vvvv,SPACE=(CYL,(1,1)),             C
//                DISP=(sss,KEEP),DCB=(,BLKSIZE=133,RECFM=F)
//MLISP DD DSNAME=J629.TRANS,UNIT=2314,                          C
                  VOLUME=SER=SYS06,DISP=(OLD, KEEP)
//LISPIN DD *
OPEN (MLISP SYSFILE INPUT)
RESTORE (MLISP)
MEXPR 0
(Mlisp program goes here)
```

Batch --

```
                                                              Col. 72


//XXXX JOB(nnnn, bin, r, ℓ),'your name',MSGLEVEL=1
//JOBLIB DO DSNAME=SYS2 .PROGLIB,DISP=OLD
//JOBLIB  EXEC  PGM=LISPA
//LISPOUT  DD  SYSOUT=A
//MLISP  DD  DSNAME=J629.TRANS,UNIT=2314,                           C
                VOLUME=SER=SYS06, DISP=(OLD,KEEP)
//LISPIN  DD  *
OPEN  (MLISP  SYSFILE  INPUT)
RESTORE (MLISP)
MEXPR ()


                (Mlisp program goes here)
```

where,

XXXX	=	job name
nnnn	=	job number
bin	=	Bin number
r	=	Run time
ℓ	=	Lines of output
pppp	=	Wylbur output file name
vvvv	=	Volume, for example, SYS06
xxx	=	Status, NEW or OLD

See Users Manual[3] for more detail on control cards. After //LISPIN you are taking to LISP/360. MEXPR() is the call on the translator.

Bibliography

1. J. McCarthy, et al. Lisp 1.5 programmers manual. M.I.T. Press, Cambridge, Mass., 1962.

2. J. Kent and R. Berns. Lisp/360 reference manual. Campus Facility Users Manual, Stanford Computation Center, 1967.

3. Users Manual, Computation Center, Stanford University, Stanford, California 94305.