MACRO Definitions for LISP

by Timothy P. Hart

Abstract

This is a recreation of MIT AI Memo 57 which was written by Timothy P. Hart on October 22 1963. The scan that exsist from MIT is in very poor shape and hard to read, an attempt has been made to keep the original formating of the text and code as close as possible to the original. The document and Timothy Hart certainly deserves something better than a bad scan, and hopefully this recreation will do that justice. The paper describes the first implementation of LISP macros in LISP 1.5, one of the most beautiful inventions in the history of programming languages.

In LISP 1.5 special forms are used for three logically separate purposes: a) to reach the alist, b) to allow functions to have an indefinite number of arguments, and c) to keep arguments from being evaluated.

New LISP interpreters can easily satisfy need (a) by making the alist a SPECIAL-type or APVAL-type entity. Uses (b) and (c) can be replaced by incorporating a MACRO instruction expander in define.

1. The property list of a macro instruction will have the indicator MACRO followed by a function of one argument, a form beginning with the macro's name and whose value will replace the original form in all function definitions.

2. The function macro[1] will define macro's just as define[1] defines functions.

3. define will be modified to make macro expansions.

Examples:

1. The existing FEXPR csetq may be recplaced by the macro definition:

```
MACRO ((
(CSETQ (LAMBDA (FORM) (LIST (QUOTE CSET)(LIST (QUOTE QUOTE)(CADR FORM))
(CADDR FORM))))
))
```

2. A new macro stash will generate the form found frequently in PROG's:

```
x := cons[form;x]
```

using the macro stash, one might write instead of the above:

```
(STASH FORM X).
```

Stash may be defined by:

```
 MACRO ((
(STASH (LAMBDA (FORM)(LIST (QUOTE SETQ)(CADAR FORM)(LIST (CONS (CADR FORM)
(CADAR FORM))) )))
))
```

3. New macros may be defined in terms of old. enter is a macro for adding a new entry to the table (dotted pairs) stored as the value of a program variable.

```
enter[form] MACRO≡ list[STASH;list[CONS;cadr[form];caddr[form]];
cadddr[form]]
```

Incidentally, use of macros will alleviate the present di(iculty resulting from the 90 LISP compiler's only knowing about those fexprs in existence at its birth.

The macro defining function macro[1] is easily defined:

```
macro[1] ≡ deflist[1;MACRO]
```

The new define is a little harder:

```
define[1] ≡ deflistc[mdef[1];EXPR]
mdef[1] ≡ [
     atom[1]->1;
     eg[car[1];QUOTE]->1;
     member[car[1];(LAMBDA LABEL PROG)]->
cons[car[1];cons[cadr[1];mdef[caddr[1]]]];
     get[car[1];MACRO]->mdef[get[car[1];MACRO]
[1]]
     T->maplist[1;λ[[j];mdef[car[j]]]]]
```

4. The macro for select illustrates the use of macros as a means of allowing functions of an arbitrary number of arguments:

```
    select[form] MACRO≡ λ[[g];
list[list[LAMBDA;list[g];cons[COND;
maplist[cadr[form];λ[[l];
  [null[cdr[l]]->list[T;car[l]];
  T->list[list[EQ;g;caar[l]];cadar[l]] ]]]
]];cadr[form]]][gensym[]]
```