A SOLUTION OF THE FUNCTIONAL ARGUMENTS PROBLEM IN LISP

by John McCarthy
Dean E. Wooldridge, Jr.

We propose to handle the functional argument problem by introducing a function called function (different from the function function in LISP 1.5) and making the following change in programs using functional arguments: rather than write "function[λ[[a_1; . . . ;a_n];<expr>]]", we write "function[[a_1; . . . ;a_n];<expr>;[s_1; . . . ;s_m]]", where the s_i are variables used in the expression <expr> (or in functions called as a result of the fact that <expr> is called) which must be evaluated at the time the functional argument is set up. "function[[a_1; . . . ;a_n];<expr>;NIL]" will be interpreted as λ[[a_1; . . . ;a_n];<expr>] and all arguments and parameters of <expr> will be evaluated at the time <expr> is called (this is consistent with the behavior of LISP before the addition of the FUNCTION-FUNARG hack).

For the interpreter:

function[[a_1; . . . ;a_n];<expr>;[s_1; . . . ;s_m]] generates and defines (Gsymbol (LAMBDA (s_1 . . . s_m) <expr>)); replaces itself (using rplaca and rplacd) with: (FUNCTION (a_1 . . . a_n) Gsymbol (s_1 . . . s_m)); and returns a pointer to: (LAMBDA (a_1 . . . a_n)(Gsymbol (QUOTE $\bar{s}_1$) . . . (QUOTE $\bar{s}_m$))), where $\bar{s}_i$ = eval[s_i] (or, in LISP 1.5, eval[s_i;$ALIST]). Subsequent calls of function recognize that the second argument is a LISP-generated symbol and behave as above except that Gsymbol is not again defined.

The compiler handles functional arguments in the following manner:

The definition of Gsymbol is compiled at compile time; and the main program is compiled with code to list the s_i and place a pointer to this list in the second word of the two-word block on the push-down list which contains the functional argument. The first word of the functional argument block is to be loaded with a transfer to Gsymbol. The arguments a_i are placed on the push-down list in the same manner as are arguments of functions which are not arguments of Gsymbol. Functional arguments which require no special treatment use only one word on the push-down list.

Consider the following example:

```
test[x;u] = if atom[x] then u[] else
     test[car[x];λ[[];test[cdr[x];u]]], where the last x (in
. . . test[cdr[x]; . . . ) is to have the value that was current at the
time test[car[x]; . . . ] was entered.  In the new notation, this function
would be written:
test[x;u] = if atom[x] then u[] else
     test[car[x];function[[];test[cdr[x];u];[x]]].  Or, if u is
permitted to be modified and the value of u at the time the functional
argument call is set up is desired, then the definition becomes:
test[x;u] = if atom[x] then u[] else
     test[car[x];function[[];test[cdr[x];u];[x;u]]].
```

To illustrate the ***FULL POWER*** of this scheme, we present:

```
testr*[x;y;f;p;u] =
     if p[x] then f[x] else
     if atom[x] then u[y] else
     testr*[car[x];y;f;p;function[[y];testr*[cdr[x];y;f;p;u];[x;p;u]]].
```

(We wish to acknowledge our indebtedness to Prof. Harold McIntosh of the Instituto Politecnico Nacional of Mexico City, whose contribution to the above example is obvious.)

Here it is assumed that y and f are constant but that p and u may not be, and, of course, x is definitely not constant. After functional argument juggling by the read routine, define, or the function function, the internal representation of this definition may be as the following S-expression:

```
(TESTR* (LAMBDA (X Y F P U)(COND
     ((P X)(F X))
     ((ATOM X)(U Y))
     (T (TESTR* (CAR X) Y F P (FUNCTION (Y)(Gsymbol X P U))))
))),
```

where Gsymbol is defined as:

```
(Gsymbol (LAMBDA* (X P U)(TESTR* (CDR X) Y F P U))),
```

and the meaning of "LAMBDA*" will be made clear shortly.

For the purpose of illustrating a way in which this scheme might be implemented by the compiler, we shall follow these conventions:

Arguments of functions are transmitted via the push-down list.

Values of functions are returned in the accumulator.

List pointers are true address pointers.

Function argument pointers are in decrement fields.

Called functions clock up and down the push-down list pointer (in index PDX).

The push-down list expands towards higher locations.

The last cell of a function's push-down block saves the return address, which the function picks up from the subroutine index (SRX).

Arguments are evaluated by the calling function.

The appearance of the push-down list at the time of execution of Gsymbol is indicated on the page following the sample code.

On the 7090, testr* might be compiled as shown on the next pages.

```
TESTR* TXI   *+1,PDX,-8
       PXA   0,SRX
       STO   0,PDX          save return address
       CLA   -7,PDX         x
       STO   1,PDX
       XEC   -4,PDX         p
       TZE   *+3            go if not[p[x]]
       XEC   -5,PDX         f  (smart compiler remembers that x is still
*                              in the right place on the push-down list)
       TRA   RETURN
*
       TSX   ATOM,SRX
       TZE   *+5            go if not[atom[x]]
       CLA   -5,PDX         y
       STO   1,PDX
       XEC   -3,PDX         u
       TRA   RETURN
*
       TSX   CAR,SRX
       STO   -1 PDX         save car[x]
       PXA   0 PDX
       PAC   0,SRX
       TXI   *+1,SRX,-7
       PXA   0,SRX          pointer to x
       TXI   *+1,SRX,3
       PXA   0,SRX          pointer to p
       STO   3 PDX
       TXI   *+1 SRX,1
       PXA   0,SRX          pointer to u
       STO   4 PDX
       CLA   =3             number of arguments of list
       STO   1,PDX
       TSX   LIST,SRX       list values of x, p, and u current at time of
*                           setting up functional argument
       STO   6,PDX
       CLA   -1 PDX         car[x] previously set aside
       STO   1,PDX
       CLA   -6 PDX         y
       STO   2,PDX
       CLA   -5,PDX         f
       STO   3,PDX
       CLA   -4,PDX         p
       STO   4,PDX
       CLA   CALL           Gsymbol
       STO   5,PDX
       TSX   TESTR*,SRX
*
RETURN XCA
       CLA   0,PDX
       PAX   0 SRX          restore return index
       XCA
       TXI   *+1 PDX,8      restore push-down list
       TRA   1,SRX          return
*
CALL   TSX   Gsymbol,SRX
```

```
Gsymbol TXI   *+1,PDX,-1
        CLA   0,SRX             pick up XEC that called Gsymbol
        STA   *+1
        LDQ   **,PDX            pick up list of x, p, and u
        TXI   *+1,PDX,-5        clock up push-down pointer
        PXA   0,SRX
        STO   0,PDX
        STQ   1,PDX
        TSX   CAR,SRX           get true pointer to saved x
        PDC   0,SRX
        CLA   0,SRX
        STO   -4,PDX            saved x is second argument of Gsymbol
        TSX   CADR,SRX          get true pointer to saved p
        PDC   0,SRX
        CLA   0,SRX
        STO   -3,PDX            saved p is third argument of Gsymbol
        TSX   CADDR,SRX         get true pointer to saved u
        PDC   0,SRX
        CLA   0,SRX
        STO   -2,PDX            saved u is fourth argument of Gsymbol
        CLA   1,SRX
        STO   -1,PDX            u takes two push-down list words
*
*    *    *    *    *    *    *    *    *    *    *    *    *
*
        CLA   -4,PDX            saved x
        STO   1,PDX
        TSX   CDR,SRX
        STO   1,PDX             cdr[x]
        CLA   -5,PDX            y
        STO   2,PDX
        CLA   -11,PDX           f from most recent entry to TESTR* -- since
*                               Gsymbol is compiled as a subcompilation of
*                               TESTR*, and can be called only by TESTR*, it
*                               knows where the arguments of TESTR* may be
*                               found on the push-down list and need not use
*                               free (special) variable mechanism
        STO   3,PDX
        CLA   -3,PDX            saved p
        STO   4,PDX
        CLA   -2,PDX            saved u, word 1
        STO   5,PDX
        CLA   -1,PDX            saved u, word 2
        STO   6,PDX
        TSX   TESTR*,SRX
        XCA
        CLA   0,PDX
        PAX   0,SRX             restore return index
        XCA
        TXI   *+1,PDX,6         restore push-down list
        TRA   1,SRX             return
```

Push-down list configuration for the execution of testr*[. . .]:

Location relative to push-down index	contents	
-7	x	
-6	y	
-5	f	(TSX f,SRX)
-4	p	(TSX p,SRX)
-3	u	(TSX u,SRX)
-2	u	(PZE **)
-1	**	(temporary storage)
0	<return>	<<<PDX points here during initial execution of testr*

(When Gsymbol is entered, the push-down list is expanded as follows.)

1	y	
2	x	(saved)
3	p	(saved)
4	u	(saved)
5	u	(saved)
6	<return>	<<<PDX points here during execution of Gsymbol

(Gsymbol sets the arguments for the next testr* call into the next five cells on the push-down list and goes to TESTR*.)

The meaning of "LAMBDA*" is now apparent: Gsymbol is formally defined as a function with one argument; but it really has four -- the last three of which it sets up for itself on the push-down list. Furthermore, since Gsymbol can only be called by testr* and it is not recursive, it really needn't take up space on the push-down list. Thus, Gsymbol might be a function of one argument which really has no arguments at all -- and which is, in fact, not even a function. The first part of Gsymbol -- down to the line of asterisks -- is the FEXPR part of the function, and might well be coded as a separate linking routine -- McCarthy's "rudimentary apply."

We have glossed over the problem of telling functions which call functionals how many push-down words the arguments of the functional use; this difficulty, we believe, may be overcome by some sort of simple modification of the calling sequence for functional arguments -- a scheme which would pack subroutine locations and list pointers into one word on the push-down list, for example.