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[\lambda[a_1; \ldots ;a_n];<expr>]]", we write "function[[a_1; \ldots ;a_n];<expr>];[s_1; \ldots ;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; \ldots ;a_n];<expr>;NIL]" will be interpreted as \[\lambda[a_1; \ldots ;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:

\[\text{function}[[a_1; \ldots ;a_n];<expr>];[s_1; \ldots ;s_m]]\] generates and defines (Gsymbol (LAMBDA (s_1 \ldots s_m) <expr>)); replaces itself (using rplaca and rplacd) with: (FUNCTION (a_1 \ldots a_n) Gsymbol (s_1 \ldots s_m)); and returns a pointer to: (LAMBDA (a_1 \ldots a_n)(Gsymbol (QUOTE s_m)) \ldots (QUOTE s_m)), where \[s_i = \text{eval}[s_i]\] (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 way:
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:

\[ \text{test}[x;u] = \text{if atom}[x] \text{ then } u[] \text{ else } \text{test}[\text{car}[x];\lambda[];\text{test}[	ext{cdr}[x];u]] \]

where the last \( x \) (in

\[ \ldots \text{test}[\text{cdr}[x];\ldots \ldots] \]

is to have the value that was current at the
time \( \text{test}[	ext{car}[x];\ldots \ldots] \) was entered. In the new notation, this function
would be written:

\[ \text{test}[x;u] = \text{if atom}[x] \text{ then } u[] \text{ else } \text{test}[	ext{car}[x];\text{function}[,]\{\text{test}[	ext{cdr}[x];u];[x]\}] \]

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:

\[ \text{test}[x;u] = \text{if atom}[x] \text{ then } u[] \text{ else } \text{test}[	ext{car}[x];\text{function}[,]\{\text{test}[	ext{cdr}[x];u];[x;u]\}] \]

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

\[ \text{testr}^*[x;y;f;p;u] = \]

\[ \text{if } p[x] \text{ then } f[x] \text{ else } \]

\[ \text{if atom}[x] \text{ then } u[y] \text{ else } \]

\[ \text{testr}^*[\text{car}[x];y;f;p;\text{function}[,]\{\text{testr}^*[\text{cdr}[x];y;f;p;u];[x;p;u]\}] \]

(We wish to acknowledge our indebtedness to Prof. Harold McIntosh of the

Instituto Nacional 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:

\[ (\text{TESTR}^* \text{ (LAMBDA} (X Y F P U) (\text{COND} (\text{P} X) (F X)) ((\text{ATOM} X) (U Y)) \text{ (T} \text{ (TESTR}^* \text{ (CAR} X) Y F P \text{ (FUNCTION} Y) (Gsymbol} (X P U))))) \]

where \( Gsymbol \) is defined as:

\[ (Gsymbol \text{ (LAMBDA}^* (X P U) \text{ (TESTR}^* \text{ (CDR} X) Y F P U))) \]

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.
save return address
x

go if not[p[x]]
p
f (smart compiler remembers that x is still in the right place on the push-down list)

go if not[atom[x]]
y
u

save car[x]
pointer to x
pointer to p
pointer to u
number of arguments of list
list values of x, p, and u current at time of setting up functional argument
car[x] previously set aside

restore return index
restore push-down list
return
Gsymbol TXI *+1,PDX,-1  
CLA 0,SRX  
STA *+1  
LBQ *+,PDX  
TXI **+,PDX,-5  
PXA 0,SRX  
STO 0,PDX  
STQ 1,PDX  
TSX CAR,SRX  
PDC 0,SRX  
CLA 0,SRX  
STO -4,PDX  
TSX CADDR,SRX  
PDC 0,SRX  
CLA 0,SRX  
STO -3,PDX  
TSX CADDR,SRX  
PDC 0,SRX  
CLA 0,SRX  
STO -2,PDX  
STO 1,PDX  
CLA -4,PDX  
CLA -5,PDX  
STO 2,PDX  
STO -11,PDX  
STO 3,PDX  
CLA -3,PDX  
STO 4,PDX  
CLA -2,PDX  
STO 5,PDX  
CLA -1,PDX  
STO 6,PDX  
TSX TESTR*,SRX  
XCA  
CLA 0,PDX  
PAX 0,SRX  
XCA  
TXI *+1,PDX,6  
TRA 1,SRX

pick up XEC that called Gsymbol  
pick up list of x, p, and u  
clock up push-down pointer  
get true pointer to saved x  
saved x is second argument of Gsymbol  
get true pointer to saved p  
saved p is third argument of Gsymbol  
get true pointer to saved u  
saved u is fourth argument of Gsymbol  
u takes two push-down list words  
saved x  
cdr[x]  
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  
saved p  
saved u, word 1  
saved u, word 2  
restore return index  
restore push-down list  
return
Push-down list configuration for the execution of testr*[ ... ]:

<table>
<thead>
<tr>
<th>Location relative to push-down index</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7</td>
<td>x</td>
</tr>
<tr>
<td>-6</td>
<td>y</td>
</tr>
<tr>
<td>-5</td>
<td>( f (\text{TSX } f,\text{SRX}) )</td>
</tr>
<tr>
<td>-4</td>
<td>( p (\text{TSX } p,\text{SRX}) )</td>
</tr>
<tr>
<td>-3</td>
<td>( u (\text{TSX } u,\text{SRX}) )</td>
</tr>
<tr>
<td>-2</td>
<td>( u (\text{PZE } *) )</td>
</tr>
<tr>
<td>-1</td>
<td>** (temporary storage)</td>
</tr>
<tr>
<td>0</td>
<td>&lt;return&gt;</td>
</tr>
</tbody>
</table>

(When Gsymbol is entered, the push-down list is \texttt{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 \texttt{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.