STANFORD ARTIFICIAL INTELLIGENCE PROJECT
Memo No.8
(Preliminary)

September 26, 1963.

STORAGE CONVENTIONS IN LISP 2

by John McCarthy

Abstract: Storage conventions and a basic set of functions for LISP 2 are proposed. Since the memo was written, a way of supplementing the features of this system with the unique storage of list structure using a hash rule for computing the address in a separate free storage area for lists has been found.

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).

At the LISP 2 meeting in July, considerable discussion was devoted to the problem of allowing a wide variety of new types of entities, but no definite conclusions were reached. In the last few weeks a new approach to types has been developed that simplifies the problem greatly. The simplicity comes from separating the problem of computation types (e.g. integer vs real) from the problem of storage types and solving the storage type problem by itself.

Here is the solution. The basic entity in the system is list structure as in LISP 1.5. However, a word with negative sign is a storage layout word and can say any of the following things:-

1. Following me in memory is a block of n words none of which have relocatable addresses or decrements.

2. Following me in memory is a block of n words each of which has relocatable address and decrement.

3. Following me is a block of n words whose relocatability is determined by bits in as many of the words that immediately follow me as is necessary (Both address and decrement relocate if either does).

4. Following me is a block of n words of program. The bits that immediately follow say which words have relocatable address parts.

In each case the address part of the layout word may contain a type name (e.g. real or integer) usable for determining computation type and serving to distinguish the type in sets defined by the direct union operation of "A Basis".

The first advantage of this scheme is that the marker of the packing garbage collector can know how to trace the storage from the layout words alone; it does not have to refer to separate type definition statements.

The second advantage of this scheme is that the public pushdown list can be made to include layout words just like free storage itself. This simplifies the marker which can now simply start marking at the head of the pushdown list.

We would like to include the LISP 2 program itself and its subroutines in the free storage area and have it referred to from the top of the pushdown list. This would mean that parts of the system could be excised, and the garbage collector would automatically pack everything else.

We shall now review the garbage collector for LISP 2 which is substantially as discussed at the July conference. It operates in four phases.

1. Marking. In a separate table two bits are reserved for each word of free storage and this table is initially set to all zeroes. The marker starts at the head of the pushdown list and traces the memory structure determined by list structure and the layout words. Each word that it finds has one of three entries made in the mark table according to whether it doesn't relocate (numbers), has address relocation only (certain program words), or has address and decrement relocatable (list structure). Layout words are of the third type.

As in LISP 1.5, marking proceeds in both directions in a list word so that all words accessible from the pushdown list are marked. Relocatable addresses in program are not traced further however.

2. Listing unmarked words. A linear sweep is made through the free storage area and every word not marked in the mark table is put into a chain. Each word of the chain has the number of unmarked words preceding it put in its address part.

3. Relocating addresses. In another linear sweep through storage each word that has a relocatable address or decrement indicated in the mark table is modified. The amount of modification is determined by looking in the location referred to and counting backwards till an unmarked word is found. This word has the amount of modification in its address part.

4. Moving. A third linear sweep moves each word the required amount.

Associated with this memory structure are a number of functions. As in LISP 1.5 we have car, cdr, and cons. Supplementing car $[x]$ and cdr $[x]$ is cwr $[x]$ (contents of the word in register) whose value is the full word contents of the register whose address is x. Words in a block specified by a layout word are obtained by functions like car $[x + 3]$ or cwr $[x + 3]$ 1.

Supplementing cons $[x; y]$ and corresponding to the first three types of layout words listed above we have the following functions:

$$\text{mk1}\left[n;\ \text{type};\ w_1;\ldots;\ w_n\right]$$

whose value is the location of a layout word of the first kind, labelled with type in its address part and followed by n words with contents w_1; ...; w_n;

$$\text{mk2}\left[n;\ \text{type};\ a_1;\ d_1;\ \ldots a_n;\ d_n\right]$$

whose value is the location of a layout word labelled with type and followed by n words whose address and decrement parts are specified by the arguments a_1, d_1, ... a_n, d_n

$$\text{mk3}\left[n;\ \text{type};\ r_1,\ldots r_1;\ \begin{matrix} w_1 \\ a_1;d_1 \end{matrix};\ \ldots\ \begin{matrix} w_n \\ a_n d_n \end{matrix}\right]$$

Here the $r_1,\ldots r_i$ specify the relocatability of the n following words and the w's or a-d pairs give the entries themselves. We do not define a mk4 function, at least for the present.

The analogues of rplaca and rplacd will exist only in the program feature which will be as like ALGOL 60 as is reasonable. One will write in the M-language

$$\text{cwr}\left[x + 3\right] :\ =\ 3.14$$

or $$\text{cwadr}\left[x + 3\right] :\ =\ 4.2$$

or $$\text{caddr}\left[x\right] :\ =\ (\text{PLUS}\quad A\quad B)$$

Besides these functions which are analogous to LISP 1.5 we also want to be able to reserve an array in storage for later assignment statements. This is accomplished by the functions:-

declare 1 [n; type]

declare 2 [n; type]

and declare 3 [n; type; r_1; ... r_i]

which have as value the location of the layout word of a new block of a given type but do not actually put anything into the block (It is set to all zeroes).

The storage scheme described above provides the ability to use most of the kinds of entity proposed at the conference without making any commitments to specific computation types.

In their present form the functions are unsafe. For example if x is the location of a layout word specifying a block of three unrelocatable words, then cwr(x +4) does not depend on the quantity represented by x but belongs in another block, and even x + 1 will not be a legitimate LISP 2 entity since it is not the location of either list structure or specification words. This suggests that these functions not be used directly, but that the compiler generate the appropriate functions when reading type definition statements. Thus the compiler might generate $\lambda[[x]; cwr[x + 1]]$ and $\lambda[[x]; cwr[x + 2]]$ to pick up the real and imaginary parts of a complex number. For the time being, however, a very powerful and easy to produce system can be made from the basic functions. Misuse of the basic functions will lead to obscure bugs because the marker will become confused.

JMcC/afg.