LISP: PROGRAM IS DATA

A HISTORICAL PERSPECTIVE ON MACLISP

Jon L White
Laboratory for Computer Science, M.I.T.*

ABSTRACT

For over 10 years, MACLISP has supported a variety of projects at M.I.T.'s Artificial Intelligence Laboratory, and the Laboratory for Computer Science (formerly Project MAC). During this time, there has been a continuing development of the MACLISP system, spurred in great measure by the needs of MACSYMA development. Herein are reported, in a mosiac, historical style, the major features of the system. For each feature discussed, an attempt will be made to mention the year of initial development, and the names of persons or projects primarily responsible for requiring, needing, or suggesting such features.

INTRODUCTION

In 1964, Greenblatt and others participated in the check-out phase of Digital Equipment Corporation's new computer, the PDP-6. This machine had a number of innovative features that were thought to be ideal for the development of a list processing system, and thus it was very appropriate that the first working program actually run on the PDP-6 was an ancestor of the current MACLISP. This early LISP was patterned after the existing PDP-1 LISP (see reference 1), and was produced by using the text editor and a mini-assembler on the PDP-1. That first PDP-6 finally found its way into M.I.T.'s Project MAC for use by the Artificial Intelligence group (the A.I. group later became the M.I.T. Artificial Intelligence Laboratory, and Project MAC became the Laboratory for Computer Science). By 1968, the PDP-6 was running the Incompatible Time-Sharing system, and was soon supplanted by the PDP-10. Today, the KL-10, an advanced version of the PDP-10, supports a variety of time sharing systems, most of which are capable of running a MACLISP.

MACSYMA (ref. 2) grew out of projects started on the 7090 LISP 1.5, namely Moses' SIN program and Martin's MATHLAB. By implementing the Project MAC Symbolic and Algebraic manipulation system in LISP, many advantages were obtained. Of particular importance were (i) a basic data convention well-suited for encoding algebraic expressions, (ii) the ability for many independent individuals to make programming contributions by adhering to the programming and data framework of LISP, and (iii) the availability of a good compiler and debugging aids in the MACLISP system. As the years rolled by, the question was asked "What price LISP"? That is, how much faster could the algebraic system be if the advantages brought by the LISP system were abandoned and an all-out effort was made in machine language? Moses has estimated that about a factor of two could be gained (private communication), but at the cost of shifting much of the project resources from mathematical research to coding and programming. However, that loss could have been much larger had not MACLISP development kept pace, being inspired by the problems observed during MACSYMA development, and the development of other projects in the A.I. Laboratory. The most precarious strain placed on the supporting LISP system by MACSYMA has been its sheer size, and this has led to new and fundamental changes to MACLISP, with more yet still in the future. Many times, the MACSYMA

*During the calendar year 1977, the author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

system was not able to utilize the solution generated for one of its problems, due to the familiar trap of having already too much code invested in some bypass solution; but there has generally been an interchange of ideas amongst those groups using MACLISP at the A.I. Lab and LCS, and another group may have received the benefit of an idea born by MACSYMA needs.

Because the system is still evolving after a decade of development, it is useful to think of it as one big piece of *data*, a *program* still amenable to further critical review and emendation. Below are presented some of the developments of this past 10 years, with a little bit of explanation as to their significance and origin.

HOW WE GOT TO WHERE WE ARE

Clever Control Features

In 1966, Greenblatt suggested abandoning the a-list model for program variables, and returning to a standard save-and-restore stack model such as might be used by a recursive FORTRAN. This was the first LISP to do so, and a later LISP developed at Bolt, Beranek, and Newman (BBN) in Cambridge used a model whereby storage for program variables was dynamically allocated on the top of a stack. Both stack models could achieve a significant speed-up over the a-list models, but at a cost of limiting the use of FUNCTION (see ref. 3). The BBN LISP later became INTERLISP (ref. 4), and currently has a stack model with the same function capabilities as the a-list model. In 1975, the PROGV feature was added and is apparently unique to MACLISP. PROGV is essentially PROG, except that the list of variables is not syntactically present, but rather is computed as an argument to PROGV; previously, about the best one could do was to call EVAL (or APPLY) with a dynamically-constructed LAMBDA expression.

In 1969, Sussman, noticing features of the MULTICS operating system, demanded some similar features for MACLISP: asynchronous interruption capability, such as alarmclocks, job-console control keys, hardware faults, interprocess communication, and exceptional process conditions (chiefly, errors). Many LISP systems now permit the user to supply functions for handling standard LISP errors, and provide for some mechanism at the job-console to interrupt the system, putting it into a top-level-like loop called BREAK. MACLISP permits interruption capability on any character of the input-console keyboard; the user may designate any function to be run when a particular key is typed. To some degree, these features appeared concurrently in INTERLISP, but especially the stackframe and debugging facilities of INTERLISP inspired similar ones in MACLISP. In mid-1976, MACLISP could finally give an interrupt to the user program on several classes of hardware-detected conditions: access (read or write) to a specific address, attempted access to non-existent address, attempted write access into read-only memory, parity error, and illegal instruction. Furthermore, some operating system conditions could trigger special interrupts: system about to shut down in a few minutes, and console screen altered by system. Evident from the development of LISP-embedded systems was the need for a NOINTERRUPT facility, which could protect user-coded processes from an accidental, mid-function aborting such as might occur during an asynchronous interrupt. Steele designed and implemented the current scheme in late 1973.

Sussman's development of MICRO-PLANNER (ref. 5) required some more capabilities for intelligent, dynamic memory management; and thus White, in 1971, introduced programmable parameters for the garbage collector — a minimum size for each space, a maximum allowable, and a figure demanding that a certain amount be reclaimed (or found free) after a collection. Then in the next year came the GC-DAEMON mechanism, whereby a user function is called immediately after each garbage collection so that it can intelligently monitor the usage of memory and purposefully modify the memory-management parameters. Baker, who has recently done work on concurrent garbage collection (ref. 6), has produced a typical storage monitor using the MACLISP mechanisms (ref. 7).

Sussman's later development of CONNIVER (ref. 8) showed the need for a sort of non-local GOTO, as a means of quickly aborting a computation (such as a pattern-matching data-base search) that had gone down a wrong path. Thus in 1972 White devised the CATCH and THROW facilities (THROW provides a quick, non-local break-out to a program spot determined by CATCH), and implemented FRETURN as a means of an impromptu "THROW" out of any stackframe higher up than the current point of computation (this is especially effective if an error break occurs, and the user can supply by hand a correct return value for some pending subroutine call several levels up the stack). In 1975, Steele coded the EVALHOOK feature, which traps each interpretive entry to EVAL during the evaluation of a piece of code; this permitted users to write debugging packages that can effectively "single-step" through an evaluation.

The embedding of advanced programming-language systems in LISP, such as MACSYMA, MICRO-PLANNER, CONNIVER, and LLOGO (ref. 9) required a means of insulating the supporting system (written as LISP code) from the users code (written in the new experimental language). Sussman and White noticed that the action of INTERN was primarily a table look-up, and they implemented this table (in 1971) as a LISP array, which array is held as the value of the global variable OBARRAY. Thus a user can change, or even LAMBDA-bind, the INTERN environment. Similarly, the action of the programmable reader could be controlled by exposing its syntax and macro table as the value of the global variable READTABLE, which was done in 1972. In 1975, the MAPATOMS function as found in INTERLISP was implemented for quickly applying a function to all the objects on a given OBARRAY. All these embedded systems wanted to have better control over the LISP top-level and break-level loops; so in 1971 two features were added: 1) ability to replace the top-level and break-level action with a form of the user's choice, and 2) a facility to capture control after a system-detected error has occurred but before re-entry to the top level. At first, the error-break permitted only exiting by quitting out back to top level, but later these breaks were such that many errors could be corrected and the computation restarted at the point just prior to the error detection. By early 1975, it was noted that many applications wanted to alter what might be called the default input reader and the default output printer; the former because their code files were written with many macro and special facilities, and the latter because of the occurrence of circular list structure. Thus the two variables READ and PRIN1, if non-NIL, hold a user-supplied function for these operations.

I/O Facilities

In 1968, White proposed a programmable, macro-character input reader, and by the summer of 1969, the reader was in operation. Since that time, some other LISPs have added certain special features to their readers, such as inputting 'A as (QUOTE A), or as in INTERLISP, permitting the user to change the meaning of break, separator, and escape characters; but to the author's knowledge none have any user-programmable macro[1] facility, nor so wide a range of parsing options as does MACLISP.

The PRINT function of MACLISP has remained relatively neglected over the years; but in 1973 Steele implemented the PRINLEVEL and PRINLENGTH facilities as inspired by the INTERLISP PRINTLEVEL facility. LISP has always had the notion of "line length", such that if more than a specified number of characters were output without an intervening newline character, the a newline was automatically inserted by the system (this was especially practical in the days when model 33 Teletypes were the main terminal used, and the operating system did not take care of preventing too long a line). MACLISP allowed an override on this automatic insertion feature, but in 1976 Steele modified this facility so that, even when not overriden, it would not insert the generated newline character in the middle of some atom. Along with the macro-reader in 1968, White installed dynamically-variable base conversion for fixnums, so that any base between 2 and 36 could be used; for what it's worth, Steele extended this for roman numerals also in 1974.

[1] Of course the macro functions are written in LISP, what else!

The problem of "perfect" output for floating-point numbers on the PDP-10 has apparently not been solved in any other system. That is, given the more-or-less standard input algorithm for base conversion from floating-point decimal numbers (dfpns) to floating-point binary numbers (bfpns), construct an output conversion algorithm such that

i) every representable bfpn is converted to a shortest dfpn, and

ii) if e is a representable bfpn, and e^* is its dfpn image by the output algorithm, then the input algorithm applied to e^* produces *exactly* e.

In 1972, White devised and installed in MACLISP an algorithm that was more nearly "perfect" than any other known to the author or to persons of his acquaintance; and in May 1977 White and Steele improved that algorithm so that they think it is "perfect" (a proof of which is forthcoming). Most other algorithms will increase the least-significant bit of some numbers when passed through the read-in of print-out cycle (see reference 10 for a possible explanation of why this problem is so hard). Golden anticipates MACSYMA's usage of this capability, "perfect" print-out, if it indeed is truly so.

Inspired by LISP 1.6 (ref. 11), a preliminary version of a multiple I/O scheme was coded up by Stallman in 1971. Prior to this, MACLISP could effectively READ from at most one file at a time, and PRINT out onto at most one file at a time; furthermore, there were no provisions for I/O other than the ASCII streams implicit in READ and PRINT. That preliminary version was abandoned in early 1973, and a decision was made to copy the design of the MULTICS version I/O (which had been developed rather independently). This scheme, coded by Steele and ready for use early in 1975, has been termed "Newio". It has since been undergoing continuing check-out and development up until now, and in January 1977 became the standard MACLISP on the ITS versions, although we have not yet made the necessary modifications to the TOPS-10 version.

Between 1967 and 1971, the A.I. Lab Vision Group, and MACSYMA Group saw the need for a faster method of getting compiled LISP subroutines off disk storage and into a running system. Back then, the compiler would produce a file of LAP code, which would be assembled in each time it was required. The first step in this direction was taken in 1969 when White devised a dynamic array space, with automatic garbage collection. Then White and others worked out a relocatable format for disk storage such that the load in time could be minimal; Steele and White implemented this scheme between 1972 and 1973, called FASLOAD. Golden reported that the time to load in all the routines comprising the then-existing MACSYMA dropped from about an hour to two minutes; continuing MACSYMA development certainly required this FASt LOADing scheme. Closely following in time was the AUTOLOAD scheme, whereby a function that was not part of the in-core environment, but resident in FASL format on disk, would be FASLOADed in upon first invocation.

Arithmetic Capabilities

Perhaps the most stunning achievement of MACLISP has been the method of arithmetic that has permitted FORTRAN-like speed from compiled LISP code. In 1968, Martin and Moses, foreseeing future needs of MACSYMA, demanded better arithmetic capabilities from MACLISP. In 1969, Martin changed the implementation of numbers so that FIXNUMs and FLONUMS consumed only one word, rather than three — that is, the LISP 1.5 format was abandoned and numbers were implemented merely as the pointer to the full-word space cell containing their value. Such a scheme had already been accomplished, partially, in other LISPs. After that change in the interpreter had been completed, some new functions were introduced for type-specific arithmetic:

for fixed point: + - * / 1+ 1-
for floating point: +$ -$ *$ /$ 1+$ 1-$
for either (but not mixed): = < >

Later, more functions were added, such as fixed-point square-root, and greatest-common-divisor. The fixed-point functions would be an automatic declaration to the compiler that all arguments and results would be fixnums, and that all arithmetic can be modulo 2^{35}; similarly, the flonum functions would specify the use of floating point hardware in the compiled code.

At the same time, Binford suggested installing separate full-word stacks for FIXNUMs and for FLONUMs, and interpreting these stack addresses as the corresponding type number. Then White proposed eliminating the discontinuity in FIXNUM representation caused by the INUM scheme, so that open-compilation of numeric code would need no extra, interpretive-like steps to extract the numerical value from a LISP number;[2] White also designed a scheme for using the number stacks, interfacing compiled subroutines with one another and with the interpreter. The redesign of number storage, and the design of a numeric subroutine interface, was for the purpose of permitting the compiler to produce code similar to what a PDP-10 FORTRAN compiler could produce on essentially numeric programs.[3] Work then began on the compiler to take advantage of all this, and a preliminary version for arithmetic code was operational by late 1971, under the care of Golden and Rosen who did most of the early coding. Rosen and White developed optimization in the compiler during 1972, and White continued this work through the end of 1976. In 1974, White and Steele extended the array data facilities of MACLISP to include FORTRAN-like arrays of fixnums and flonums so that the compiler could optimize array references in numerical code; see Steele's paper describing the current output available from the compiler (ref. 13).

Early along in MACSYMA development, Moses and Martin saw the need for variable-precision integer arithmetic, and thus the BIGNUM functions were born, with most algorithms taken from Knuth (ref. 14). During 1972 and 1973, Golden suggested the need in MACSYMA for some of the usual transcendental functions, like SIN, COS, natural logarithm and anti-logarithm, and arc-tangent (these were adapted from some rational approximations originally developed by White in 1967); for GCD, HAULONG, HAIPART, and improvements to the the exponentiation function EXPT; and for the ZUNDERFLOW switch, which permits interpretive arithmetic routines to substitute a real zero for any floating-point result that causes a floating-point underflow condition. By combining the binary and Lehmer algorithms from Knuth (ref. 15), Gosper produced a GCD algorithm early in 1976 which runs much faster on bignum inputs. Also, in 1976, a feature was added to the interpretive floating-point addition and subtraction routines such that if the sum is significantly less than the principal summand, then the sum is converted to zero; the variable ZFUZZ holds a scale-factor for this feature, which is still considered experimental (LISP370 has a more pervasive use of a similar feature in all floating-point arithmetic and I/O functions).

Randomness has always been a property of MACLISP, having had a linear-shift-register RANDOM number generator since early times. This generator produced a maximally-long sequence, was extremely fast, and moderately acceptable for most applications. However, it failed the correlated-triples test, and when it was used to generate random scenes for display on the LOGO Advent color projector, it produced some very nice kaleidoscopic pictures; so in late 1976, a modification of Knuth's Algorithm A (ref. 16) was coded by Horn.

Ancillary Packages

A number of ancillary functions have been coded in LISP, mostly by persons who were LISP users rather than system developers, and are kept stored in their compiled, FASL format for loading in when desired. In 1970, Binford coded a small, but powerful, subset of the INTERLISP in-core editor as a LISP package, but this was later recoded in machine language; a more extensive version of the INTERLISP editor has been coded by Gabriel in 1975. In 1970, Winston designed and coded INDEX,

[2] MACLISP, by inspecting the numerical value of a number coming into the FIXNUM-conser, supplies a canonical, read-only copy for fixnums in the range of about -1000. to +2000. This significantly reduces the number of new cells required by running arithmetic code, without significantly slowing down the operations. Currently, no similar action is taken for FLONUMs.

[3] The generally-accepted opinion in 1968, and indeed in some quarters up until 1973, was that LISP is *inherently* a hundred times slower on arithmetic than is FORTRAN. Fateman's note in 1973 effectively rebutted this opinion (ref. 12), but in 1969 it took faith to go ahead with this plan; only Martin and the author had a clear resolve to do so then.

a package to analyze a file of LISP programs and report on certain properties therein. During 1972, Goldstein replaced an existing, slow pretty-printer (called GRIND) with a programmable pretty-printer (ref. 17), and Steele spruced-up an existing TRACE package to have more features. After the Newio scheme became operational, two packages were coded for the fast dumping onto disk and retrieval therefrom of numeric arrays, and a FASDUMP package was implemented for MACSYMA that could quickly and efficiently store list structure on disk (Kulp had a hand in developing this package, but it may no longer be in use). Many of these user-supplied packages now reside on a disk area called LIBLSP, which includes a FORMAT package by White for printing out numbers under control of a format (such as is used in FORTRAN), a package for reading and printing circular list structures, various debugging packages and s-expression editors, and many others.

In 1973 Pratt was continuing work on a "front end" for LISP, CGOL (ref. 18), which he had begun at Stanford University in 1971, and he had it generally operational at a number of sites by 1975. It exemplifies the Pratt operator-precedence parser (now used at the front end of MACSYMA), and has some of the character of MLISP (ref. 19). However, the CGOL-to-MACLISP conversion is dynamic and fast, and furthermore, an acceptable inverse operation has been implemented, so that one can effectively use this ALGOL-like language while still retaining all the advantages of MACLISP (fast interpreter, good compiler, many debugging aids, etc.). It is not at all impractical to replace the MACLISP default reader and printer with CGOL's (see notes on READ and PRIN1 in the last paragraph of "Clever Control Features" above), so that CGOL may be properly thought of as an alternate external syntax for LISP. See reference 7 for a practical example — one particular GC-DAEMON function for MACLISP, coded in CGOL.

MIDAS, the A.I. Lab's assembly-language system for the PDP-10, cooperates with MACLISP to the extent of being able to produce a FASL format file. A number of these ancillary packages have thus been coded in machine language for greater efficiency. In mid 1973, Steele coded a version of Quicksort (ref 20) which is autoloadable as the function SORT; in 1976, after Newio became stable, Steele coded a file-directory query package (called ALLFILES), and designed a package for creating and controlling subjobs (tasks) in the ITS time-sharing environment (called HUMBLE). Using the HUMBLE package, Kulp and others interfaced the text editor TECO with MACLISP, for increased programmer efficiency in debugging and updating LISP programs. Kulp and others had proposed a text-processing system suitable for use with a photo-composer to be written in MACLISP and using these features, but this has not yet been realized. With the ALARMCLOCK facility for periodic interrupts, and HUMBLE for driving sub-tasks, MACLISP is fully equipped for becoming a time-sharing system.

Export Systems

Martin's desire to be able to use MACSYMA on the MULTICS system led to the start of a MULTICS version of MACLISP, begun in late 1971 by Reed; after this was fully operational in 1973, Moon, who had worked on it wrote the now-extinct MACLISP Reference Manual published in March 1974 (ref. 21). Although there has been little use of MACSYMA on the MULTICS version, it was successfully transplanted there; several other extension systems developed on the PDP-10 version were also successfully tested on the MULTICS version, such as LLOGO and CONNIVER.

In the summer of 1973, the MACLISP system was extended to permit its use on TOPS-10, DEC's non-paged time sharing system. Much help on this development has come from members of the Worcester Polytech Computation Center, and from the resources of the Computer Science department of Carnegie-Mellon University. The impetus for having a TOPS-10 version came from many academic institutions, where students with interests in artificial intelligence had been intrigued by MICRO-PLANNER and CONNIVER and their applications, and had wanted to experiment with these systems on their own PDP-10s. Later, as M.I.T. graduate students and professors moved to other universities, they took with them the desire to use MACLISP, rather than any of the other available LISP alternatives. The major difficulty in export to these other institutions has been their lack of adequate amounts of main memory — few places could even run the MACLISP compiler, which requires 65^{+}K. At one

time Moses had a desire to export MACSYMA through this means, but this has not proved feasible. Even for the KI-10 and KL-10 processors, which have paging boxes, the TOPS-10 operating system does not give user programs sufficient control over the page-map; consequently, this version of MACLISP is to some degree less efficient in its memory utilization.

The TENEX and TOPS-20 operating systems should be able to support the TOPS-10 version of MACLISP, under a compatibility mode, but there has been some difficulty there. In 1971, a specially tailored version of MACLISP was run under the TENEX system, but this version died out for lack of interest. If future interest demands it, there should be no trouble in getting almost the full range of MACLISP features found on the ITS version to be implemented in a TOPS-20/TENEX version. In 1976 Gabriel adapted the TOPS-10 version to run on the Stanford A.I. Laboratory operating system, and there is currently an increasing body of users out there.

Revised Data Representations

A major step was taken in 1973 when the long-awaited plans to revise the storage strategy of MACLISP saw the light. A plan called Bibop (acronym for BIg Bag Of Pages), inspired in part by the prior INTERLISP format, was designed by White, Steele, and Macrakis; and this was coded by Steele during the succeeding year. The new format relieves the need for a LISP user to make precise allocations of computer memory, and permits dynamic expansion of each data space (although only the array storage area can be dynamically reduced in size). In 1974, numeric arrays were added, and in 1976 a new data type called HUNK was added as a s-expression vector without any of the overhead associated with the array data type. Steele's paper in these proceedings (ref. 22) gives a detailed account of how the current storage picture looks inside MACLISP.

Especially MACSYMA, as well as Winograd's SHRDLU and Hewitt's PLASMA systems, needed the efficiency and versatility of these new formats. The concept of "pure free storage" entered the picture after Bibop became operational: this is list and s-expression structure that is essentially constant, and which can be removed from the active storage areas that the garbage collector manages. Furthermore, it can be made read-only, and shared among users of the same system; in MACSYMA, there are myriads of such cells, and the consequent savings is enormous. Thus the incremental amount of memory required for another MACSYMA user on the system starts at only about 45K words!

The Compiler

Greenblatt and others wrote a compiler for the PDP-6 lisp, patterned initially after the one for 7090 LISP on CTSS. This early attempt is the grandfather of both the current MACLISP and current LISP 1.6 compilers. However, optimizing LISP code for the the PDP-6 (and PDP-10) is a much more difficult task than it might first appear to be, because of the multiple opportunities provided by the machine architecture. That early compiler had too many bugs to be really useful, but it did provide a good, basic structure on which White began in 1969 (joined by Golden in 1970) to work out the plans for the fast-arithmetic schemes (see ref. 13). The LISP 1.6 compiler has apparently not had so thorough a check-out and debugging as the MACLISP compiler, since its reputation is unreliability. The INTERLISP compiler was produced independently, and seems to be quite reliable; but comparisons have shown that average programs compile into almost twice as many instructions through it than through the MACLISP compiler.

Ad-Hoc Hacs

As the number of new and interactive features grew, there was observed need for a systematic way to query and change the status of various of the operating system and LISP system facilities. We did not want to have to introduce a new LISP primitive function for every such feature (there are scores!), so thus was born in 1969 the STATUS and SSTATUS series. The first argument to these functions selects one of many operations, ranging from getting the time of day from a home-built clock, to reading the phase of the moon, and to setting up a special TV terminal line to monitor the garbage collector. Later,

in 1975, the function SYSCALL was added as a LISP entry into the time-sharing system's CALL series of operations. (See reference 23 for information on the ITS system.)

Between 1970 and 1972, the demands of the A.I. Lab Vision group necessitated the installation of a simulated TV camera, called the FAKETV, along with a library file of disk-stored scene images. A cooperative effort between the Vision group and the LOGO group led to the design of a Display-slave — a higher, display-orientated language for use with the Lab's 340 Display unit using the PDP-6 as an off-line display processor. Goldstein, because of his interest in LLOGO (ref. 9), participated in the initial design along with Lerman and White; the programming and coding were done by the latter two.

In 1973, terminal-input echo processing (rubout capability) was enhanced, and cursor control was made available to the user for the existing display terminals. When the A.I. Lab began using the home-built TV terminal system, Lieberman coded a general-purpose display packages in LISP for use on the TV display buffer. When Newio became available in 1975, Lieberman and Steele showed examples of split-screen layouts usable from LISP, and in 1976 Steele showed how to code a variety of "rubout" processors in LISP. Furthermore, Newio permitted extended (12-bit) input from the keyboards associated with these terminals.

In 1973, MACLISP copied a feature from LISP 1.6 for improving facilities in linkage between compiled subroutines — the UUOLINKS technique. All compiled subroutine calls are done indirect through a table, which contains interpretive links for subroutine-to-subroutine transfer. Under user option, these links may be "snapped" during run time — that is, converted to a single PDP-10 subroutine transfer instruction. A read-only copy is made of this table (after a system such as MACSYMA is generated) so that it may be restored to its unsnapped state at any time. The advantage of this is that, normally, subroutine transfers will take place in one or two instruction executions, but if it is desired to debug some already compiled subroutines, then one need only restore the interpretive links from the read-only copy.

Inspired by MACSYMA's history variables, MACLISP adopted the convention in early 1971 that the variable "*" would hold the most recent quantity obtained at top level.

In 1973, White coded an s-expression hashing algorithm called SXHASH, which has been useful to routines doing canonicalization of list structure (by hashing, one can greatly speed-up the search to determine whether or not there is an s-expression copy in a table EQUAL to a given s-expression).

To accommodate the group that translated the lunar rocks query-information system from INTERLISP to MACLISP, the convention was established in 1974 that car[NIL]=cdr[NIL]=NIL. This seems to have been widely accepted, since it simplifies many predicates of the form (AND X (CDR X) (CDDR X)) into something like (CDDR X).

WHERE DO WE GO FROM HERE?

The major problem now with MACLISP, especially as far as MACSYMA is concerned, is the limitation imposed by the PDP-10 architecture — an 18.-bit address space, which after overhead is taken out, only leaves about 180K words for data and compiled programs. Steele discusses some of our current thinking on what to do about this in his paper (ref. 22) of these proceedings, under the section "The Address Space Problem". Since the LISP machine of Greenblatt (ref. 24) is such an attractive alternative, and is even operational now in 1977, we will no doubt explore the possibilities of incorporating into PDP-10 MACLISP some of its unique features, and in general try to reduce the differences between them. For the future of MACSYMA, we foresee the need for new, primitive data types for efficient use of complex numbers and of double-precision floating-point numbers. We anticipate also the need to have a version efficiently planted in the TOPS-20 system.

FULL NAMES OF PERSONS ASSOCIATED WITH MACLISP DEVELOPMENT AND MENTIONED IN THIS PAPER

MIT Professors	Research Staff	Students
Joel Moses	Jon L White	Guy L. Steele Jr.
William A. Martin	Jeffrey P. Golden	David A. Moon
Gerald J. Sussman	Richard Greenblatt	Eric C. Rosen*
Ira P. Goldstein	Thomas O. Binford*	John L. Kulp
Vaughan Pratt	Jerry B. Lerman*	Richard P. Gabriel*
Patrick H. Winston	R. William Gosper*	Henry Lieberman
Terry L. Winograd*		Richard M. Stallman
Carl E. Hewitt		Stavros Macrakis
Richard J. Fateman*		David P. Reed
Berthold K. P. Horn		Henry G. Baker, Jr.

* = No longer at M.I.T.

REFERENCES

1. Deutsch, L., and Berkeley, E.; The LISP Implementation for the PDP-1 Computer, in *THE PROGRAMMING LANGUAGE "LISP"*. edited by Berekeley, E., and Bobrow, D., Information International Inc., 1964.
2. *MACSYMA Reference Manual,* Project MAC Mathlab Group, M.I.T., November 1975.
3. Moses, J.; *The Function of FUNCTION in LISP.* AI Memo 199, Artificial Intelligence Lab, M.I.T., June 1970.
4. Teitelman, W.; *INTERLISP Reference Manual (Revised edition).* Xerox Palo Alto Research Center, 1975
5. Sussman, G., Winograd, T, and Charniak, E.; *Micro-Planner Reference Manual (revised).* AI Memo 203A, Artificial Intelligence Lab, M.I.T., December 1971.
6. Baker, H.; A Note on the Optimal Allocation of Spaces in MACLISP. Working Paper 142, Artificial Intelligence Lab, M.I.T., March 1977.
7. Baker, H.; List Processing in Real Time on a Serial Computer. Working Paper 139, Artificial Intelligence Lab, M.I.T., February 1977.
8. McDermott, D., and Sussman, G.; *THE CONNIVER REFERENCE MANUAL.* AI Memo 259A, Artificial Intelligence Lab, M.I.T., January 1974.
9. Goldstein, I.; *LLOGO: An Implementation of LOGO in LISP.* AI Memo 307, Artificial Intelligence Lab, M.I.T., June 1974.
10. Matula, D.; In-and-Out Conversions. *CACM 11, 1,* January 1968, pp. 47-50.
11. Quam, L.; *STANFORD LISP 1.6 MANUAL.* SAILON 28.3, Artificial Intelligence Lab, Stanford University, 1969.
12. Fateman, R.; "Reply to an Editorial", *SIGSAM Bulletin,* 25, March 1973, pp. 9-11.
13. Steele, G.; Fast Arithmetic in MACLISP. Proceedings of the 1977 MACSYMA Users Conference, NASA CP-2012, 1977. (Paper no. 22 of this compilation).
14. Knuth, D.; *The Art of Computer Programming,* V2. Addison-Wesley, 1969, pp. 229-240.
15. ———, *ibid.*, pp. 293-307.
16. ———, *ibid.*, pp. 26-27.
17. Goldstein, I.; *Pretty-Printing, Converting List to Linear Structure.* AI Memo 279, Artificial Intelligence Lab, M.I.T., Feburary 1973.
18. Pratt, V.; CGOL — An Alternative External Representation for LISP Users. Working Paper 121, Artificial Intelligence Lab, M.I.T., March 1976.
19. Smith, D.; *MLISP.* AIM-135, Artificial Intelligence Lab, Stanford University, 1970.
20. Knuth, D.; *The Art of Computer Programming.* V3, Addison-Wesley, 1973, pp. 114-116.
21. Moon, D.; *MACLISP Reference Manual, Revision 0.* Laboratory for Computer Science (formerly Project MAC), M.I.T., March 1974.
22. Steele, G.; Data Representations in PDP-10 MACLISP. Proceedings of the 1977 MACSYMA Users Conference, NASA CP-2012, 1977. (Paper no. 21 of this compilation).
23. Eastlake, D.; *ITS Status Report.* AI Memo 238, Artificial Intelligence Lab, M.I.T., April 1972.
24. Greenblatt, R.; The Lisp Machine. Working Paper 79, Artificial Intelligence Lab, M.I.T., November 1974.