STANFORD ARTIFICIAL INTELLIGENCE PROJECT
Memo No.13

February 20, 1964.

THE NEW LISP SYSTEM (LISP 1.55)

by Dean E. Wooldridge

Abstract: The new LISP system is described. Although differing only slightly, it is thought to be an improvement on the old system.

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

THE NEW LISP SYSTEM (LISP 1.55)

The new LISP system (LISP 1.55) differs somewhat from the old system, but only in (generally) inessential details. It is hoped that the changes will be widely hailed as improvements.

Verbos and the Garbage Collector

The garbage collector now prints its message in a single-spaced format; thus, the amount of paper generated by a program with many cons'es is substantially less than formerly. Furthermore, the garbage collector printout may be suspended by executing verbos [NIL]; and the printout may be reinstated by executing verbos [*T*].

Flap Trap

Every now and then a state of affairs known as floating-point trap occurs - this results when a floating-point arithmetic instruction generates a number whose exponent is too large in magnitude for the eight-bit field reserved for it. When this trap occurs and the offending exponent is negative, the obvious thing to do is to call the result zero. The old system, however, simply printed out a "flap trap" error message and went on to the next part of the program. The new system stores a floating-point zero in the accumulator when an underflow occurs. (There has, as yet, been no request to have "infinity" stored in the accumulator when an overflow occurs.)

Time, et al.

The original LISP system (c 1958) had the unfortunate habit of printing, when evalquote was entered or exited (at the beginning of a packet - just after the TEST, TST, DEBUG, SET, or SETSET card was read - and at the end of a packet - just before the overlord card following STOP)) . . .)) was read), "The time has come, the walrus said, . . ." (see the LISP 1.5 Manual, page 51), with the date and time inserted therein. The reaction of the new LISP programmer to this was a polite, but insincere, smile; whereas the more experienced LISP people had considerable difficulty tolerating it politely. Stanford's versions of LISP have, wisely, avoided using this printout.

However, the new system does print the time upon entering and leaving evalquote. In fact, two times are printed, but in a neat, concise, impersonal manner which, it is felt, is more suitable in the "Age of Automation" than the quote from Lewis Carroll. The times are printed in minutes and milliseconds; the first time is the age of the packet - by definition, this is zero when evalquote is first entered - and the second time is the age of the system being used. Thus, when evalquote is exited, the time printout tells how much time was spent in the execution of the packet, and how much time has been spent in execution of SET or SETSET packets since the birth of the system plus the time spent in the most recent packet.

It is also possible to determine how much time is required to execute a given function. Timel[] initializes two time cells to zero and prints out, in the same format as the evalquote printout, two times, and these are both zero. Time[] prints (again in the evalquote time printout format) the time since the last execution of time[] and the time since the last execution of timel[]. The use of the time and timel functions has no effect on the times recorded by evalquote.

LAP and Symtab

Heretofore, LAP has not only returned the symbol table as its value, but printed it out as well. This phenomenon is familiar to those who have had much at all to do with LAP or the compiler. The LAP in the new system always printedthe function name and the octal location in which the first word of the assembled function is stored (if the assembled function is not a SUBR or FSUBR, then only the octal origin of the assembledcode is printed). The printout is left-justified on the utput page and has the form: "FUNCTION*NAME (ORIGINAL xxxxxQ)".

The value of LAP is still the symbol table, but the printing of the symbol table may be suspended by executing symtab[NIL]; and the printing may be restored by executing symtab[*T*].

Non-printing Compiler

The problem of the verbosity of the compiler is only slightly abated by the symtab function. The remainder of the trouble may be cured by executing listing[NIL]. This turns off the printout of the LAP code generated by the compiler. And, of course, the printout may be reinstated by executing listing[*T*]. Thus, for a perfectly quiet (except for the ORIGIN printout by LAP), one need only execute symtab[NIL] and listing[NIL] before compiling.

Tracecount (Alarm-Clock Trace)

The trace feature of LISP is quite useful, but, with very little encouragement, it can generate waste-baskets full of useless output. Often a programmer will find that his output (without tracing) consists of many lines of garbage collector printout, an error message, and a few cryptic remarks concerning the condition of the push-down list at the time the error occurred. In such a situation, one wishes he could begin tracing only a short time before the occurrence of the error. The tracecount function permits exactly this. Tracecount[x] causes tracing (of those functions designated to be traced by the trace function, as before) to begin after x number of function entrances. Furthermore, when the tracecount feature has been activated, by execution of tracecount[x], the blank space in the garbage collector printout will be used to output the number of functions entered at the time of that garbage collection, each time the arguments of value of a traced function are printed the number of function entrances will also be printed, and if an error occurs, the number of function entranced accomplished before the error occurred will be printed.

The tracecount feature (or Alarm-Clock Trace as it is called by Professor Marvin Minsky of M.I.T.) enables a programmer to run a job (preceding the program by "tracecount[0]"), estimate the number of function entrances which occur before the program generates an error condition or a wrong answer, and run the job again, tracing only the pertinent portion of the execution (a low line count is recommended for this purpose, as the tracecount feature will not turn off tracing once it has been started - the untrace function must be used to accomplish this).

Space and Eject

A small amount of additional control over the form of the data printed by LISP has been provided in the space and eject functions.

Space[*T*] causes all output to be double-spaced. Space[NIL] restores the spacing to its original status; in particular, the output of the print routine reverts to single-spacing, and the "END OF EVALQUOTE OPERATOR." printout again ejects the page before printing.

Eject[] causes a blank line with a carriage control character of 1 to be printed. The result is a skip to the top of the next page of output.

Esoteric Functions and Modifications

The following descriptions will be of interest only to that small band of LISP users consisting of people with reserved areas on the disk file.

SAVE. A new overlord directive has been included in the system which makes it possible to save the current status of core and the active registers on a specified section of the disk in the event that the time estimate is exceeded. SAVE (beginning in column 8) before a TEST, TST, DEBUG, SET, or SETSET card will cause a clock trap to produce a transfer to a portion of overlord which saves the current values of various registers and the location of the instruction which would have been executed next if the clock trap had not occurred, and stores the system on the section of the disk specified by the most recent DISK directive. Then control returns to the monitor and program is halted.

In order to restart the saved program, it is only necessary to load a No.2 card with LISPx, where x is the number of the area of the disk on which the saved program is stored, in the system name column. (Of course, if the job is to be run by the Computation Center, a #1 card is also necessary. If the job is to be loaded on-line, then a blank card is necessary behind the #2 card - one-card programs are not seen by the on-line card-to-tape routine). The save feature makes it possible to run a long job in short chunks, allowing the programmer to look at intermediate results before spending more computer time on the program, but allowing the program to take up where it left off, rather than having to repeat computation previously done.

The save feature is only active during the packet immediately following the SAVE card. Thus, if several packets are included in the program, several SAVE cards (one for each packet) must be used in order to be sure of saving the system.

UNTIME. This routine is not available to the programmer, but its mention here may prevent some anxiety. In the event that the program time estimate is exceeded during disk I/O, using the old system, one finds himself in the position of having half of one system and half of another stored in core on on the disk. This situation would be intolerable if the programmer were trying to save some definitions on the disk so that he could use them later, or if the save feature were active (so that the saved system would not be quite what it should be). To avoid this unpleasantness, the disk I/O routines have been modified so that the clock is, in essence, turned off during disk reading or writing, and two seconds automatically added to the elapsed time at the conclusion of the read or write operation. A clock trap that would normally have occurred during the execution of the read or write will be executed before the I/O can take place.

A few programmers with very large programs have long bemoaned the inability, in LISP, to communicate between systems stored on different sections of the disk. The functions tape, rewind, mprint, and mread have been designed to alleviate this difficulty. Tape[s], where s is a list, allows the user specify up to ten scratch tapes; if more than ten are specified, only the first ten are used. The value of tape is its argument. The initial tape settings are, from one to ten, A4, A5, A6, A7, A8, B2, B3, B4, B5, B6. The tapes must be specified by the octal number which occurs in the address portion of a FAP instruction to rewind that tape; that is, a four-digit octal number is required - the first (high-order) digit is a 1 if channel A is desired, 2 if channel B is desired; the second digit must be a 2; the third and fourth are the octal representation of the unit number. Thus, to specify that scratch tapes one, two, and three are to be tapes A4, B1, and A5, respectively, execute tape [(1204Q, 2201Q,1205Q)]. Only the low-order fifteen bits of the numbers in the tape list are used by the tape routines, so it is possible to use decimal integers or floating-point numbers in the tape list without generating errors.

REWIND. Rewind[x] rewinds scratch tape x, as specified in the most recently executed tape function.

MPRINT. Mprint[x;s] prints the S-expression s on scratch tape number x. The format of the output is identical to the normal LISP output (it is, in fact, generated by the print routine), and is suitable for printing. Note that if scratch tape 1 has been designated to be A2, then mprint[1;s] has exactly the same effect as print[s]. The value of mprint is the list printed.

MREAD. Mread[x] reads one S-expression from scratch tape x. The value of mread is the expression read. Again, if scratch tape 1 corresponds to tape unit A3, then mread is identical, from the programmer's point of view, with read.