STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

File Copy

September 1970

SRI - TRACE PACKAGE FOR PDP-10 LISP

by

Robert E. Kling

Artificial Intelligence Group
Technical Note 37

(Supersedes TN 27)

SRI Project 8259

This research was sponsored by the Advanced Research Projects Agency of the Department of Defense and was monitored by the National Aeronautics and Space Administration under Contract NAS12-2221.

I've written a new LISP TRACE package which supercedes the current TRACE package. Briefly, it differs from the basic LISP trace package by: (a) having a clearer, indented printout format; (b) allowing a variety of information to be suppressed or printed when a function is traced; (c) allowing a user to break a function or to suppress the tracing when a particular condition is met; (d) modifying edit[ℓ] and grindef[ℓ] to be compatible with traced functions; and (e) has a PRINTLEVEL control. To some extent these features are easier to use than to describe. I suppose that most users will select their personal preferred subset of those available.

1. All the functions described below are upwards compatible with any that are used in the system-associated TRACE package.

2. To access the package, type N when the LISP system asks whether you want TRACE?. Then type (INC SYS: SRITRF) from within LISP. The new package uses about 4300 free words, compared to 1800 used by the original package. So allow 2.5K more free sotrage (or expect 2.5K less free space available). A list of functions utilized in this package is bound to the atom SRITRF.

3a. All the printing done by the system is effected by a global variable, PRINTLEVEL, which is initially set to 10.

3b. When the TRACE package encounters an error or if a function is broken, the system calls help[m] which prints a message m and then enters a READ-EVAL-PRINTN loop. printn[s], the print function, is responsive to PRINTLEVEL.

3c. This system sports a new format which is indented for various levels of function calls and variable bindings. A global parameter named %-INDENT (which is initialized to 0) controls the left margin position.

4a. To trace $fn_1 \ldots fn_k$, execute (TRACE FN1 FN2 ... FNK) as usual. All the arguments will be printed. Consider a function $fn_2[x;y;z]$. Suppose you only want to see the values of x and cdr(y); and you want only to see the length of the value of $fn_2[x;y;z]$. Then execute

```
(TRACE (FN2 X (CDR Y) ; : (LENGTH FN2)))    .
```

4b. The preceding example provided an instance of the general trace format which is:

$$\text{Trace}[fn_1;\ fn_2;\ \ldots\ fn_k] \quad ,$$

where

$$fn_j = \begin{cases} \text{FNJ} \\ \text{or (FNJ } \ell_1) \\ \text{or (FNJ } \ell_1;\ \ell_2\text{:}\quad g(fn_j)) \end{cases} \quad .$$

FNJ is the name of the function to be traced. In the list format, ℓ_1 is a list of variables or functions whose values are to be printed just before fn_j is evaluated (e.g. just as it is entered). ℓ_2 is a list of functions or variables to be printed just after fn_j is exited. $g(fn_j)$ represents some function of the value of fn_j which will be printed instead of the value of fn_j. If ℓ_1 is empty, a header will be printed when fn_j is entered, but no argument bindings will be printed. If ℓ_2 is empty, no special functions or arguments will be printed after fn_j is exited. If $g[fn_j]$ is empty, the value of fn_j itself will be returned. If $g[fn_j]$ is specified, the name of fn_j should be inserted as a token for its value. Thus,

```
(LENGTH FNJ)

(CONS (CAR FNJ) (CADR FNJ))

(TESTFN FNJ T) etc.
```

In the non-default conditions, the elements of ℓ_1 and ℓ_2, and $g(fn_j)$ are passed onto EVAL. Thus, any LISP expression computable by EVAL at the appropriate location may be evaluated at this time. The ℓ_2 device, called POSTEVAL(uation), is quite convenient for checking the status of global parameters and data structures that may be modified by fn_j.

If fn_j = FNJ, then all the arguments of FNJ are printed when FNJ is entered and its true value is printed when it is exited. If $fn_j = \left(\text{FNJ } \ell_1;\ \ell_2;\ g[fn_j]\right)$ so that ℓ_2, <u>or</u> $g(fn_j)$ are specified, the ";" and ":" must both appear in proper order in the trace format.

4c. If a function FN3 to be traced is compiled, the TRACE routine will respond with

(FN3 IS A COMPILED FUNCTION: ARGS = ?) .

Enter a list of function arguments. Thus, to trace subst[x;y;z], type (X Y Z) when asked for an argument list.

4d. To untrace $fn_1 \ldots fn_k$ execute (UNTRACE FN1 FN2 ... FNK). When a function is traced its name is added to the global variable ALLTR. (UNTRACE ALLTR) will untrace everything.

4e. The functions which are called by the TRACE package in tracing a function may not themselves be traced. These functions include: MAPC, MAPCAR, LENGTH, SUB1, ADD1, PLUS, *DIF, SUBST, CAR, CDR, GET, GETL, and CADADR.

4f. The tracing functions are heavily error protected with errset[s]. When an error occurs during tracing the message (HELP CONTROL) is printed and you have access to EVAL with variable bindings as saved local to the error location. In order to uplevel to the bindings as they are stored at the time the last traced function was entered, type (ERR). Successive evaluations of (ERR) will unwind you step-by-step through each level of traced-function calls. A control G will bring you to the top-level eval. After exiting from an error-interrupted trace, execute (RESET) to reintialize variable bindings and restore certain global tracing parameters. Warning! Do not evaluate (RESET) within the TRACE package, but hit control G and exit to the top level first.

5a. To trace fn only when it is called by $fn_1 \ldots fn_k$, execute

(TRACEIN FN FN1 ... FNK) .

For example, to trace memq[x;ℓ] only when it is called by testfn[] and testfn2[] execute

(TRACEIN MEMQ TESTFN1 TESTFN2) .

If you want to see only x and length (ℓ), execute

(TRACEIN(MEMQ X (LENGTH L); :) TESTFN1 TESTFN2) .

5b. To deactivate the tracein feature, for MEMQ, execute

(UNTRACEIN MEMQ) .

To still trace memq[x,ℓ] within testfn2[], execute

(UNTRACEIN (MEMQ TESTFN1)) .

6a. If you only want to see a function's trace print when some predicate p is satisfied, then execute trshow[fn; p] for each function-predicate pair. For memq[x;ℓ] above,

(TRSHOW MEMQ (LESSP(LENGTH L) 10))

will only show a trace of MEMQ if length [ℓ] < 10.

6b. trunshow[fn_1;fn_2;fn_3...] reverses the effects to trshow [fn; p] for fn_1, fn_2, fn_3 ...

7a. I've written a simple break feature that stops the system when a specified function is entered and calls the HELP program. Executing break[fn; p] will halt fn upon entering if p is true. A message (FN BROKEN) is printed and the user has access to EVAL with the PRINTN feature. A broken function is halted just after its arguments are bound to the LAMBDA variables and are evaluated. To exit from a break, type OK.

7b. Unbreak[fn_1;fn_2;...] will unbreak the listed functions.

7c. All the functions which are broken are stored on a list bound to ALLBRK.

8a. tracet[ℓ] has been modified in several ways:

(1) The output format for SET-SETQ tracing is of the form:

var ← value; e.g. x ← 3.

(2) GET, GETL, REMPROP, and PUTPROP are traced along with SET and SETQ. Get[nam; prop] prints out as: PROP(NAM) = VALUE. If either <u>prop</u> or <u>nam</u> are on the list ALLSET the preceding printout will occur.

(3) The tracing may be turned off without having to reinsert list of atoms to be traced.

8b. (TRACET) will start the SET-SETQ-GET... tracing.

(TRACET A1 A2 A3 ...) will trace each of A1, A2, A3...

(TRACET T) will turn the tracing on if it has been turned off.

(UNTRACET) destroys the tracing list ALLSET <u>and</u> turns off the tracing system.

(UNTRACET A1 A2 ...) untraces A1, A2 ...

(UNTRACET T) suspends tracing printouts but does not destroy the reference list.

8c. Tracet[ℓ] will now allow printing some function of a traced variable setting instead of the actual setting itself. If settings of X1 are traced and you want to see g(x) printed instead of the value of X1, execute (TRACET(X(GX1))). The SET-SETQ-GET printing will then print out (G X1) ← <u>value</u>. This feature is particularly handy for following the growth of long lists.

9. If a function is aborted by a LISP error during tracing, type %-WHERE to find your location, e.g. (FN3 EVALUATION), etc. Otherwise, the actual break location may be confusing. Remember, all

your bindings are intact in case of error troubles!

10a. I've modified edit to work with traced functions. The SMILE function Edit[ℓ] has been modified to allow a user to change an arbitrary number of variables at once. Now one may execute:

```
(EDIT FN3 X X1 Y Y1 Z Z1)
```

to substitute X1, Y1 and Z1 for X, Y, and Z respectively. Previously this required three commands.

10b. Grindef[ℓ] has been modified to work with traced functions. In the past, grindef[fn] would printout the definition of trace[ℓ] as well as the definition of fn. Now it works properly.

10c. When performing I/O, be sure that variable tracing is turned off. Otherwise, a variable used by your program that is identical in name to a traced variable e.g. x, ℓ, etc. will print out on tape or disk a value of x whenever it is bound.

11. The present system is available in symbolic form and compiling has not yet been attempted. A user who wants listings for compilation purposes or his own use will find a complete list of functions bound to the atom SRITRF.

12. All these features are mutually compatible. No doubt hidden bugs are still lurking within the code. I'd appreciate a printout associated with any errors. Or, more effectively, SAVE your system at the time a peculiar error occurs and I'll be able to debug it quickly.

13. I'd appreciate any comments or suggestions regarding the ease or difficulty of using this system.