March 18, 1969

A LISP DEBUGGING SYSTEM

By L. H. Quam

Interactive debugging systems with breakpoints have historically been limited to assembly language programming. The most familiar of these is DDT, developed originally for the PDP-1. The development of such systems for higher level languages such as LISP is greatly simplified if one assumes that programs (functions) are interpreted. The reason for this is that the breakpoint system must modify the original function definitions, and modification of a compiled program is much more difficult than of an interpreted program.

As with other debugging systems, the LISP debugging system allows the user to capture control of his program at specified points(called breakpoints) during its execution. Having captured control, the user may examine the state of variables in his program, modify functions, change breakpoints, evaluate expressions, and if desired, continue its execution.

The main feature of the debugging system is the ability to set breakpoints around S-expresions in interpreted functions. To create a breakpoint, the user must specify to the breakpoint editor the function to edit {use (EDBRK <function name>)} and which S-expression to put the breakpoint around. The breakpoint editor has commands which allow the user to move a pointer around within his function and to search for S-expressions. When the pointer has been positioned to the desired S-expression, the B command will create a breakpoint around it by replacing the original S-expression with:

(BREAKPT B# n S-expression)

A breakpoint is activated when the LISP interpreter evaluates the breakpoint, ie., the form described above. When a breakpoint is activated, the breakpoint system evaluates a condition (predicate) associated with the breakpoint which if true specifies that the breakpoint is to be interactive. A non-interactive breakpoint activation returns the value of the S-expression as if no breakpoint had been there.

An interactive breakpoint activation allows the user to examine the state of PROG, LAMBDA, and global variables, create and destroy breakpoints, edit functions, and evaluate S-expressions.

BREAKPOINT EDITOR COMMANDS

In the following section, n will refer to a positive integer, s will refer to an S-expression, and q will refer to the S-expression currently pointed to.

n> RIGHT Move the pointer n S-expressions to the right.

n< LEFT Move the pointer n S-expressions to the left.

nV DOWN Move the pointer down n levels in parenthesis.
Eg., 1V sets the pointer to the CAR of q, the S-expression pointed to.

n^ UP Move the pointer up n levels in parenthesis.
Eg., 1^ sets the pointer to the list which contains q as a top level element.

In the <,>,^, and V commands n=1 is assumed if n is not specified.

Ss SEARCH Search to the right of the pointer for the first occurrence of the S-expression s.

B BREAKPOINT Construct a breakpoint around q, the S-expression pointed to. The S-expression must be one which the interpreter can evaluate. That is, breakpoints are not permitted around PROG labels, PROG and LAMBDA variable lists, and function names. B initializes the the breakpoint condition to T, and the variable list to the LAMBDA and PROG variables surrounding the S-expression.

Is INSERT Insert s before q.

Xs EXTEND s after q.

nD DELETE Delete n S expressions following the pointer.

Rs REPLACE Replace q by s.

P PROCEED Proceed from the current breakpoint, or exit from EDBRK, the breakpoint editor.

Es EVALUATE Evaluate (and print) s

nVs VARIABLES Set the variable list for breakpoint n to s. This is the list of variables which are printed when a breakpoint is activated into interactive mode.

nCs CONDITION Set the condition for breakpoint n to s. This condition determines whether a breakpoint becomes interactive.

nK KILL Kill (remove) breakpoint n.

In the V, C, and K commands, if n is not specified, then the current breakpoint is assumed.

W WHERE Print the S-expression pointed to.

BREAKPOINT SYSTEM OUTPUTS

To reduce the volume of output by the breakpoint system, all S-expressions are printed to only a few parenthesis levels in depth. This depth is determined by the variable D%PLEV. A non-atomic S-expression at depth D%PLEV is printed as &.

Example with D%PLEV = 2:

(LAMBDA (X) (CONS 1 (CONS X NIL)))

will print as:

(LAMBDA (X) (CONS 1 &))

When the breakpoint system is entered, if the breakpoint condition evaluates to true, then the interactive breakpoint system is entered, and the following information is printed:

A) With a teletype (or a display and without the LISP display package)

1) The location of the breakpoint is printed as follows:

***YOU ARE IN <name of function>

2) Global variables (determined by the list D%GVL) are printed in the form:

<variable name> = <value>

3) LAMBDA and PROG variables (elements of the variable list associated with the breakpoint) are printed as above.

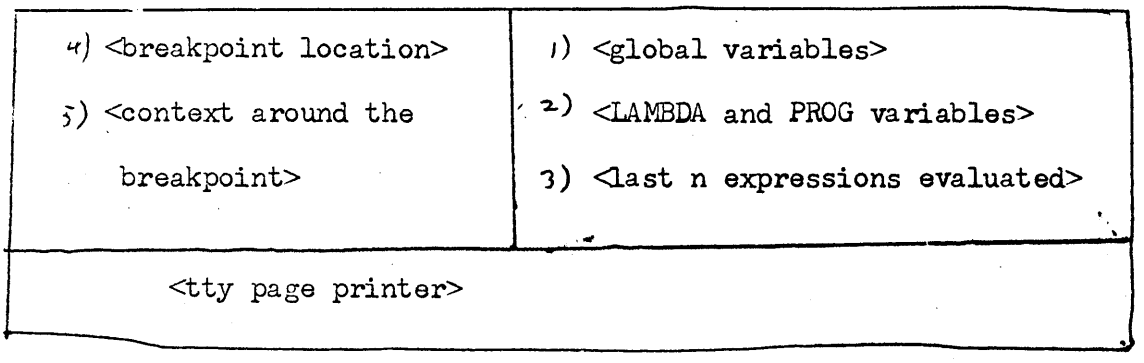
4) The S-expression currently pointed to by the breakpoint editor is printed. (When the breakpoint system is entered, the pointer is initialized to point to the breakpoint.)

5) Whenever the E, and P commands are used, the following is printed:

<S-expression> = <value>

6) Whenever the <, >, ∧, ∨, B, K, S, D, I, and R commands are used, the resulting S-expression pointed to is printed.

B) With a III display and the LISP display package (LISPDP), the screen is partitioned as follows:



1) Global variables are printed as in A2.

2) LAMBDA and PROG variables are printed as in A3.

3) The last few (determined by the variable D%MVLL) S-expressions evaluated by the E and P commands are printed as in A5.

4) The location of the breakpoint is printed in the following form:

***YOU ARE AT <name of function> <name of breakpoint>

5) The immediate context of the breakpoint (the list which contains the S-expression pointed to as a top level element) is displayed as follows:

```
(   <1st element>
    <2nd element>

 -> <S-expression pointed to>

    <last element> )
```

6) Whenever the E command is used, parts 1,2, and 3 are regenerated.

7) Whenever the <, >, ^, v, B, K, S, D, I, X, and R commands are used, part 5 is regenerated.

USE OF THE BREAKPOINT SYSTEM

To load the breakpoint system into a core image do: ALLOW 2000_8 words in BINARY PROGRAM SPACE

(INC (INPUT SYS: LAP (DEBUG.LAP)))

If using the displays, load the LISP display package, LLSPDP.

The following functions are the only top level functions needed by the user:

(BKINIT) Initialize the breakpoint system. This must be evaluated after the breakpoint system is loaded.

(EDBRK <name of function>) Enter the breakpoint editor to construct breakpoints in the specified function. The P command exits from EDBRK.

(UNBREAK . <list of function names>) Remove all breakpoints from all specified functions.