MULTILISP DEBUGGING ENVIRONMENT

by

William S. Havens

Computer Sciences Technical Report #410

December 1980

```
**********************************************
*                                            *
*     Multilisp Debugging Environment        *
*                                            *
*                   by                       *
*                                            *
*            William S. Havens               *
*                                            *
*           Revised December 1980            *
*                                            *
**********************************************
```

Department of Computer Science
The University of Wisconsin
Madison, Wisconsin 53706

Introduction:

The debugging environment for Multilisp is modelled after the Interlisp break package (Teitelman, 1974). It is also similar to the debug system of MTS/LISP (Hall, 1972). It provides interactive capabilities for error handling and algorithm debugging plus the addition from Multilisp of a single-step evaluation mode.

Once a break has been entered, the user can interrogate the state of the LISP, modify variable bindings on the stack, edit function definitions, continue the evaluation, single-step the interpreter through the evaluation of a form, restart the evaluation at some higher level on the control or access stacks, or return to top-level.

The system makes use of four special variables which can be accessed by the user:

@form: The form which caused the break to be entered called the breakform.

@frame: The control frame at the time the break was acknowledged.

@stack: Initially set to the value of @frame. Can be changed via stack searching commands.

@value: Initially @undef@. If @form is evaluated during the break, then @value is set to its value.

Entry into the System:

The debug system is entered in one of the following ways:

1: An error occurs.
2: The system is called explicitly on a form via debug.
3: The function break is called.
4: A breakpoint is encountered in a user-defined function.

Whenever an error occurs, an error message is printed, the form being evaluated at the time of the error is printed, and the break is entered.

The debug system can be called explicitly via the following two functions:

(debug <form>) type=noeval

Calls the debug system explicitly on <form>.

(break <flag> <mess>*) type=eval

This function facilitates tracing program execution and handling user defined error conditions. It first evaluates and prints each form <mess> on a single line. If the global switch @break is true or if <flag> is true, then the debug system is entered with break as the Breakform. Otherwise, break returns nil.

Finally, the debug system is entered whenever a user defined breakpoint in an interpreted function is encountered. The name of the function broken is printed and the first form within the function body becomes the breakform @form. Breakpoints are set and removed from functions with the following operations:

(breakf <foo> <pred>) type=noeval

Sets a breakpoint on the first form within the function definition of <foo>. Function definitions must be either lambda, nlambda, or qlambda-expressions. When the breakpoint is encountered, <pred> is evaluated which defaults to T. If the value of <pred> is non-NIL, the break is entered. Otherwise, it is ignored. Note that breakf actually modifies the function definition for <foo>.

(unbreakf <foo>) type=noeval

Removes an existing breakpoint from the function <foo>. If no arguments are supplied, unbreakf removes all current global breakpoints. The atom @broken-fns contains a list of all functions currently containing breakpoints.

Debug Commands:

A summary of the commands recognized by the system follows:

args Abbreviation: none

Prints the argument names and current values of the function being evaluated at @stack.

eval Abbreviation: e
Evaluates @form and prints its value. @value is set to this value.

bk <n> Abbreviation: none
Prints a backtrace of forms on the stack starting at @stack for length <n> which defaults to 1Ø.

bka <n> Abbreviation: none
Prints a backtrace of frames on the access stack starting at @stack for length <n> which defaults to 1Ø.

bkc <n> Abbreviation: none
Prints a backtrace of frames on the control stack starting at @stack for length <n> which defaults to 1Ø.

pp Abbreviation: none
Pretty-prints the form contained in the frame at @stack.

top Abbreviation: none
Resets @stack to @frame which is always the top of the stack.

find <loc> Abbreviation: f
Searches either the control or access stacks beginning at @stack looking for a locator <loc>. If found, @stack is set to that frame. <Loc> can be specified as a positive integer, negative integer, or the name of some function. <Loc> defaults to -1. Negative values, -<n>, cause @stack to be advanced <n>-frames up the control stack. Positive values for <n> cause @stack to be advanced <n>-frames up the access stack. If <loc> is specified as a literal atom, the control stack is searched for a frame created for a function of that name. Else the message ">> Not Found" is printed and @stack remains unchanged.

go Abbreviation: none
Breaks on the form at the current value of @stack.

return <form> Abbreviation: ret
Evaluates <form> and returns it as the value of the break.

restart <form> Abbreviation: res
Restarts computation from where @stack points using <form>. If <form> is not coded, computation is restarted using the previous form on the stack at @stack.

continue Abbreviation: c

Continues with @form. If @form has been previously evaluated by the eval command, it will not be re-evaluated.

step Abbreviation: s

Single-steps the interpreter causing a break on the next non-atomic form encountered by eval.

next Abbreviation: n

Evaluates @form and breaks on the next non-atomic form. If @form has been previously evaluated via the eval command, it will not be re-evaluated.

up <n> Abbreviation: |

Causes the debug system to ascend <n> levels. If there is no higher break level, control is returned to top-level. <n> defaults to 1.

stop Abbreviation: nil ||

Causes a return to top-level Multilisp.

findvar <var> Abbreviation: fvar

Finds frame containing <var> on control chain. @stack is set to this frame.

value <var> Abbreviation: v

Returns current value of <var> from access link starting at @stack, else error.

Any form typed at debug other than the above commands or there abbreviations will be evaled and its value printed.

References:

TEITELMAN, W.(1974) INTERLISP Reference Manual, Xerox Palo Alto Research Center, Palo Alto, Calif.

HALL, W.(1972) A LISP Interactive Programming Environment, M.Sc. Thesis, Dept. of Comp. Science, U. of British Columbia, Vancouver, Canada.

Comments