REVERSIBLE EXECUTION AS A DIAGNOSTIC TOOL

(Preliminary Draft)

by

Marvin Zelkowitz

Technical Report

No. 71-92

January 1971

Department of Computer Science
Cornell University
Upson Hall
Ithaca, New York 14850

REVERSIBLE EXECUTION AS A DIAGNOSTIC TOOL

(Preliminary Draft)

Marvin Zelkowitz

By the addition of only two new statements to the PL/I language, a programming system has been implemented which allows the user to retrace or backup execution of his program. He can backup N statements, to a label, to the last time that a variable was assigned a value, or until a given logical expression becomes true. Uses of this facility include the debugging of programs and applications where a stack is employed.

The first addition was a backup mechanism via a RETRACE statement which has the format RETRACE <CONDITION> AND <PL/I STATEMENT>, and has the effect that execution is retraced until <CONDITION> becomes true, the <PL/I STATEMENT> is executed, and then the program is re-executed from that point onwards.

The second change is a generalization of the PL/I ON statement by the inclusion of generalized logical expressions. The syntax of the ON statement was modified to be ON <EXPRESSION> <PL/I BEGIN BLOCK>. The user could activate an error routine by coding ON <EXPRESSION> CALL ERRORROUTINE, and <u>whenever</u> <EXPRESSION> was true, the ON unit would be activated.

With these two additions a powerful debugging facility emerges. After the error occurs the programmer can try to fix the source of the error rather than the effect. Less checks must be made while the program is executing in the forward direction. This is especially true in an interactive environment. At the very least the programmer could write a debugging routine that would "trace the N statements before an error is detected". Thus information local to the point of the error is produced without the production of the huge volumes of output typical with a tracing system.

Another use for this facility would be wherever a stack is used. Rather than including a push down stack within the program, the user could use the retrace mechanism to backup to the point before the entry was added in order to 'pop' the stack. This would be useful in the areas of symbol manipulation and compiler writing.

This system was implemented as part of the PL/I compiler called PL/C, and while it has not had customer usage, test results on randomly selected PL/C programs seem to show that performance is degraded by only a factor of two and that program size increased 40%. Execution times are significantly less than the usual times for interpretive debugging systems. Additional methods are displayed which can be used to microprogram this system in hardware. As hardware, the overhead should drop considerably, thus it would be possible to leave the debugging code within the program throughout its productive life in case an error should arise in the future.

The results show that such a facility is not very expensive to include as part of a compiler, and that future compilers would greatly benefit by the inclusion of such a debugging system. In many situations the power that such a debugging facility adds offsets the increase in execution time.

Example 1.

This program computes the roots of a quadratic equation; however, there is a keypunching error in statement 12 where an E appears instead of a C. The program executes until statement 12 which activates the error mechanism. This in turn activates the tracing mechanism which traces the remainder of the output.

Example 2.

The retrace statement has uses wherever a stack is employed. This example produces the left list of a tree by using the retrace statement to back up to a higher level in the tree when necessary. This type of routine is quite common in parsing algorithms such as in the first pass of a compiler. It is interesting since the backup mechanism is <u>not</u> being used solely for debugging in this case.

Example 3.

A binary search routine is exhibited which almost works since the end conditions are not tested properly. The backup mechanism insures that the program will continue processing with some reasonable assumptions being made. This enables the user to test all of his data, even if some of the values are incorrect. In this example, the second search (KEY =25) runs off the end of the table. The retrace insures that a reasonable value is used.

Example 4.

This example shows another non-debugging use of the retrace statement. Many times one wants to "undo" some code, such as removing an entry from a hash table. This is a difficult task; however, by use of the retrace statement to keep track of the various pointers, it becomes much easier to implement.

Please note that this system was implemented in version 4 of PL/C which did not include ON units; therefore, the logical conditional was implemented as an IF statement. Current plans are to install this system in version 5 where it will operate as described herein.

REFERENCES

1. Zelkowitz, M., Reversible Execution as a Diagnostic Tool, PhD Thesis, Cornell University, 1971

2. PL/C: The Cornell Compiler for PL/I, Department of Computer Science, Cornell University, 1970

```
A= 1.00000E+00          B= 4.00000E+00          C= 3.00000E+00;
****** IN STMT   12  ERROR  EXH7 E HAS NOT BEEN INITIALIZED. IT IS SET TO 0.

#IFI00021 ACTIVE AT ST # 0012
ERROR OCCURRED
#RETRACE(00051 ACTIVE
# AT ST # 0005 TO ST # 0011
B= 4.000000E+00;
ROUT1=-1.000000E+00;
# AT ST # 0012
****** IN STMT   12  ERROR  EXH7 E HAS NOT BEEN INITIALIZED. IT IS SET TO 0.

#IFI00021 ACTIVE AT ST # 0012
ERROR OCCURRED
#RETRACE(00051 ACTIVE
# RETRACE FAILS
B= 4.000000E+00;
ROUT2=-4.000000E+00;
# AT ST # 0013
ROUTS ARE              -1.000000E+00          -4.000000E+00
# AT ST # 0014
IN STMT   14  PROGRAM RETURNS FROM MAIN PROCEDURE
```

```
*PL/C ('DEBUG')

 /* TEST PROGRAM 1 */                                                    PL/C-R420015  01/09/71 10:16 PAGE

 STMT LEVEL NEST BLOCK      SOURCE STATEMENT                                                    ID FIELD

                          /* TEST PROGRAM 1 */
   1                      P: PROCEDURE OPTIONS(MAIN);

   2     1       1              #IF ERROR THEN DO;
   4     1    1    1                  PUT SKIP LIST('ERROR OCCURRED');
   5     1    1    1                  #RETRACE(DEBUG)
   6     1    1    1                      STATEMENTS(3) AND #TRACE PRINT;
   8     1    1    1                  END;

                          /* PROGRAM COMPUTES ROOTS OF EQUATION AX**2+BX+C=0 BY QUADRATIC  */
                          /* FORMULA */
   9     1       1        /* READ IN DATA */ GET LIST(A,B,C);
  10     1       1        /* PRINT DATA */ PUT DATA(A,B,C);
                          /* COMPUTE ROOTS */
  11     1       1              ROOT1= (-B+SQRT(B*B-4*A*C))/(2*A);
  12     1       1              ROOT2=(-B-SQRT(B*B-4*A*E))/(2*A); /*NOTE ERROR E */
  13     1       1              PUT SKIP LIST('ROOTS ARE',ROOT1,ROOT2);

                          /* THE DATA CARD CONTAINS THE VALUES 1,4,3 */

  14     1       1              END P;

ERROR  CGOC NO FILE SPECIFIED. SYSIN/SYSPRINT ASSUMED.
```

```
                                        NEXT NODE IN TREE =          1
                                        NEXT NODE IN TREE =          4
                                        NEXT NODE IN TREE =          3
                                        NEXT NODE IN TREE =          5
#RETRACE(00191 ACTIVE
# AT ST # 0019 TO ST # 0011
                                        BACKED UP TO NODE            3
#RETRACE(0019) ACTIVE
# AT ST # 0019 TO ST # 0011
                                        BACKED UP TO NODE            4
                                        NEXT NODE IN TREE =          2
#RETRACE(00191 ACTIVE
# AT ST # 0019 TO ST # 0017
                                        BACKED UP TO NODE            2
#RETRACE(00191 ACTIVE
# AT ST # 0019 TO ST # 0011
IN STMT   23  PROGRAM IS STOPPED
```

```
/* PROGRAM 2 - PRINT A TREE IN LEFT LIST ORDER */                    PL/C-96200'S  C:/.../... 12:00 PAGE

 STMT LEVEL NEST BLOCK      SOURCE STATEMENT                                                    ID FIELD

                            /* PROGRAM 2 - PRINT A TREE IN LEFT LIST ORDER */
     1                      TREEP: PROCEDURE OPTIONS(MAIN);

     2      1       1       DECLARE 1 TREE,
                                2 SON(20) FIXED BINARY,
                                2 BROTHER(20) FIXED BINARY;
                            /* READ IN TREE, EACH NODE CONTAINS A SON AND */
                            /* BROTHER POINTER */
     3      1       1           GET LIST(NUMBER_OF_NODES);
                            /* READ IN TREE */
     4      1       1           GET LIST((SON(I),BROTHER(I) DO I=1 TO NUMBER_OF_NODES));
                            /* ROOT OF TREE IS AT NODE 1 */
     5      1       1           J=1; PUSH1: ;
                            /* AT NODE J, MOVE TO SON OF J */
     7      1       1           L1: PUT SKIP LIST(' ',' ','NEXT NODE IN TREE = ',J);
     8      1       1           I=SON(J);
                            /* IF NOT AT LEAF CONTINUE DOWN STRUCTURE */
     9      1       1           IF I>0 THEN DO;
    11      1   1   1           PUSH2: J=I; GO TO L1; END;
                            /* GO TO BROTHER */
    14      1       1           L2: J=BROTHER(J);
                            /* IF NOT LEAF CONTINUE */
    15      1       1           IF J>0 THEN DO;
    17      1   1   1                   PUSH3: GO TO L1; END;
                            /* BACKUP TO FATHER POINTER */
    19      1       1           #RETRACE(DEBUG) TO (PUSH1,PUSH2,PUSH3) AND GO TO NEXT_BROT;
                            /* SEE IF FINISHED */
    22      1       1       NEXT_BROT:
    23      1       1           IF J=1 THEN STOP;
                            /* SET TO REDO THIS OPTION NEXT TIME RETRACE IS CALLED */
    24      1       1           DEBUG=0;
                            /* PRINT CURRENT NODE */
    25      1       1           PUT SKIP LIST(' ',' ','BACKED UP TO NODE',J);
                            /* GO TO BROTHER */
    26      1       1           GO TO L2;

                            /*DATA IS 5 4 0 0 0 5 0 3 2 0 0 */
                            /* WHICH IS TREE <1>-<4>-<3>-<5> */
                            /*              -<2>             */
    27      1       1       END TREEP;
```

```
****** IN STMT   15  ERROR  EXB7 N HAS NOT BEEN INITIALIZED. IT IS SET TO 0.
****** IN STMT   15  ERROR  EXB7 NEXT_ITEM HAS NOT BEEN INITIALIZED. IT IS SET TO 0.

KEY=   4                          1
KEY=  25
#IFI0005I ACTIVE AT ST # 0039
#RETRACCI0005I ACTIVE
# AT ST # 0005 TO ST # 0037                                  1
KEY=  15                          1
KEY=  -1
  1  0
  4  1
  7  0
 10  0
 15  1
 16  0
 18  0
 19  1
 21  0
IN STMT   22  PROGRAM IS STOPPED
```

```
                    /*TEST PROGRAM 3 */
1                   BINSPCH: PROCEDURE OPTIONS(MAIN);
                    /* THIS PROGRAM DOES A BINARY SEARCH OF ITEMS IN A TABLE. */
                    /* WHEN FOUND IT INCREMENTS A COUNTER. THIS WOULD BE USEFUL */
                    /* IN A SYSTEM THAT KEPT TRACK OF THE NUMBER OF ACCESSES */

                    /* DECLARE VARIABLES */
2    1        1            DECLARE I TABLE(20),
                                 ( 2 KEY, 2 ACCESS ) FIXED BINARY,
                                  DIFFERENCE FIXED BINARY;

                    /* SET ERROR EXIT IF SUBSCRIPT RUNS OFF TABLE */
3    1        1      #IF NEXT_ITEM>N THEN DO;
5    1    1    1            #RETRACE(DEBUG) CONDITION(NEXT_ITEM <=N) AND CALL FND; END;
9    1        1      #IF NEXT_ITEM <0 THEN DO;
11   1    1    1            #RETRACE(DEBUG) CONDITION(NEXT_ITEM>0) AND CALL FND; END;

15   1        1            ACCESS=0;
                    /* READ IN TABLE */
16   1        1            GET LIST(N, (KEY(I) DO I=1 TO N));
                    /* DATA READ IN IS: 9 1 4 7 10 15 16 18 19 21 */

                    /* READ IN DESIRED LOCATION */
17   1        1            LOOP: GET LIST(NEW_KEY); PUT SKIP EDIT('KEY=',NEW_KEY)
                                                                  (A,F(4));
19   1        1      /* QUIT IF FINISHED */ IF NEW_KEY <0 THEN DO;
21   1    1    1                PUT EDIT((KEY(I),ACCESS(I) DO I=1 TO N))
22   1    1    1                     (SKIP, 2 F(3)); STOP ; END;
                    /*SEARCH TABLE */
24   1        1            NEXT_ITEM=(N+1)/2; DIFFERENCE=NEXT_ITEM;
26   1        1            SRCH: IF KEY(NEXT_ITEM)=NEW_KEY THEN GO TO FOUND;
28   1        1                  IF KEY(NEXT_ITEM)>NEW_KEY THEN GO TO UPPER;
30   1        1                                                  ELSE GO TO LOWER;

                    /*FOUND ITEM IN LIST */
31   1        1            FOUND: ACCESS(NEXT_ITEM)=ACCESS(NEXT_ITEM)+1;
32   1        1                  PUT LIST(ACCESS(NEXT_ITEM)); GO TO LOOP;
                    /* TOO FAR IN TABLE, BACKUP */
34   1        1            UPPER: DIFFERENCE=(DIFFERENCE+1)/2;
35   1        1                  NEXT_ITEM=NEXT_ITEM-DIFFERENCE; GO TO SRCH;
                    /* NOT FAR ENOUGH IN TABLE */
37   1        1            LOWER: DIFFERENCE=(DIFFERENCE+1)/2;
38   1        1                  NEXT_ITEM=NEXT_ITEM+DIFFERENCE; GO TO SRCH;

                    /* REDO TRACE OPTION NEXT TIME */
40   1        1      FND: PROCEDURE;
41   2        2            DEBUG=0; GO TO FOUND; END FND;
44   1        1      END BINSPCH;
```

```
ERROR=                    4
#RETRACE(0018) ACTIVE
# AT ST # 0018 TO ST # 0006
ERROR=                    6
#RETRACE(0018) ACTIVE
# AT ST # 0018 TO ST # 0006
TABLE CONTENTS
    1   2   3   5   7   8   9  10
IN STMT   11  PROGRAM IS STOPPED
```

```
*PL/C I'DEBUG'I

C2Q /* TEST PROGRAM 4 */                                                    PL/C-R420015  01/10/71  14:32 PAGE

   STMT LEVEL NEST BLOCK      SOURCE STATEMENT                                                    IO FIELD

                              /* TEST PROGRAM 4 */
      1                       P: PROCEDURE OPTIONS(MAIN);
                              /* THIS IS AN EXAMPLE WHERE RETRACE IS USED TO UNDO A COMPLICATED */
                              /* SECTION OF CODE, SUCH AS THE DELETION OF ENTRIES IN A HASH TABLE*/

      2     1         1       DECLARE TABLE(50) FIXED BINARY, TOTAL FIXED BINARY, A(11) FIXED BINARY;
      3     1         1                 TOTAL=0; IND=0;
                              /* READ INPUT DATA */
      5     1         1                 GET LIST((A(I) DO I=1 TO 11));
                              /* READ NEXT ENTRY INTO N */
      6     1         1                 LOOP: IND=IND+1; N=A(IND);
                              /* IF N IS NEGATIVE, QUIT */
      8     1         1                 IF N<0 THEN DO;
     10     1    1    1                         PUT SKIP EDIT('TABLE CONTENTS',(TABLE(I) DO I =
                                                   1 TO TOTAL)((A,SKIP, (TOTAL) F(5)));
     11     1    1    1                         STOP; END;
                              /* INSERT N INTO TABLE (SUCH AS HASH TABLE ) */
     13     1         1                 CALL INSERT(N);
                              /* COMPUTE SOME FUNCTION BASED UPON N */
     14     1         1                 B=F(N);
                              /* DELETE ENTRY IF B FAILS SOME CONDITION */
     15     1         1                         IF (B=16) | (B=36) THEN DO;
     17     1    1    1                         PUT SKIP LIST('ERROR=',N);
     18     1    1    1                         %RETRACE(DEBUG) TO (LOOP) AND CALL BACKUP; END;
     22     1         1                 GO TO LOOP;
                              /* BACKED UP PROCEDURE CALL */
     23     1         1       BACKUP: PROCEDURE ;
     24     2         2                 /* SET DEBUG OPTION */      DEBUG=0;
     25     2         2                 /* REREAD BAD DATA */       IND=IND+1;
     26     2         2                 /* GET NEXT DATA ITEM */    GO TO LOOP; END BACKUP ;

                              /*INSERT ROUTINE - THIS CAN BE AS COMPLICATED AS DESIRED */
     28     1         1       INSERT:PROCEDURE(N); TOTAL=TOTAL+1; TABLE(TOTAL)=N; END INSERT;

                              /* FUNCTION COMPUTATION */
     32     1         1        F: PROCEDURE(N); RETURN(N*N); END F;

                              /* DATA CARDS ARE 1 2 3 4 5 6 7 8 9 10 -1 */
     35     1         1        END P;
```