Short Communications
Programming Languages

Reversible Execution

M.V. Zelkowitz
University of Maryland

Key Words and Phrases: debugging, PL/I, reversible execution, backtracking; CR Categories: 4.22, 4.42

The ability to backtrack, or retrace, the execution of a computer program has gained wider acceptance recently as a desired feature within a programming language. This is particularly useful in two different applications: (1) In debugging systems where the trace output is saved and can be interrogated under programmer control [1, 3]; (2) In artificial intelligence applications where one is trying to prove a certain result. It is frequently necessary to backup the proof and try some alternative path [2].

However, none of these systems previously developed gives the user full control over the backtracking functions. To test this possibility, the following modification was made to the Cornell PL/I compiler, PL/C [4].

Backtracking is accomplished via execution of the statement

RETRACE option;

where

option ::= TO(label) AND PL/I-statement
| STATEMENTS(expression) AND PL/I-statement
| CONDITION(expression) AND PL/I-statement

Execution of RETRACE will cause the program to backup until the 'option' is satisfied. The corresponding PL/I statement (which may be an entire BEGIN block) is then executed, and normal execution resumes. The TO option causes backup until the statement labeled 'label' is encountered; STATEMENTS causes 'expression' statements to be backed up; while CONDITION causes retracing until the logical 'expression' becomes true. For a complete description of the RETRACE feature, see [5].

Figure 1 is an example where the RETRACE feature is used to implement a tree walking algorithm. In this case, every time a leaf node is processed and there is no brother to that node, then the program finds the father node by backtracking to the last point at which the father node was known. While this example is relatively simple, the same process can be used to implement a large class of top down algorithms.

Another use of the RETRACE is in debugging systems.

Research performed at Department of Computer Science, Cornell University. Author's address: Department of Computer Science, University of Maryland, College Park, MD 20742.

Fig. 1. A tree-walking algorithm using the RETRACE facility.

```
DECLARE 1 TREE(100),
  2 SON, 2 BROTHER;
I = ROOT_NODE;
/* LOOK FOR LEAF */
SCAN_TO_LEAF: DO WHILE (SON(I) ≠ 0);
  NEW_NODE: I = SON(I);
  END;
/* PROCESS LEAF */
CALL PROCESS (TREE(I));
/* LOOK FOR BROTHER NODE */
NEXT_FATHER: IF BROTHER(I) = 0 THEN
  DO; /* GO TO FATHER */
    RETRACE TO (NEW_NODE) AND
      BEGIN; /* SEE IF DONE */
      IF I = ROOT_NODE THEN STOP;
      GO TO NEXT_FATHER;
      END;
  END;
ELSE DO; /* FOUND A BROTHER */
    I = BROTHER(I);
    GO TO SCAN_TO_LEAF;
  END;
```

Table I. Execution Statistics

STMT	SIZE N	SIZE D	TIME N	TIME D
13	504	712	0.45	0.78
12	560	880	0.95	1.34
8	320	472	0.12	0.14
15	928	1512	4.10	7.28
54	3160	4832	0.21	0.35
131	5428	7484	0.20	0.26
81	3840	5000	0.11	0.19
AVG	2106	2985	0.88	1.48

STMT—program size (statements)
SIZE N—program size (bytes)
SIZE D—program size with debugging code
TIME N—execution time (sec)
TIME D—execution time with debugging code

A user could include the RETRACE within an ON unit which is activated by some error condition. The program could backup several statements and turn on a trace of the program. In this case the user would get a trace of his program near the point of the error without the large volume of output usually associated with tracing routines. The implementation of this system involved the creation of an in-core trace table. The assignment operator was modified so that all assignments resulted in entries being made into this table. Procedure activation and termination were also modified so that proper environments could be saved.

In terms of efficiency, Table I demonstrates these results. Program size grew about 40 percent while execution time grew by a factor of 2. Since core is relatively inexpensive, and the usual interpretive debugging systems run at a factor of 40 to 50 slower, the 40 percent increase in size, and 70 percent increase in execution times are tolerable costs for these features.

Received May 1972; revised April 1973

References

1. Balzer R.M. EXDAMS: Extendable debugging and monitoring system. Proc. AFIPS 1969 SJCC Vol. 34, AFIPS Press, Montvale, N.J., pp. 567–580.
2. Feldman J.A., et al. Recent developments in SAIL—an Algol based language for artificial intelligence. AFIPS 1972 FJCC Vol. 41, AFIPS Press, Montvale, N.J., pp. 1193–1202.
3. Grishman R. The debugging system AIDS. AFIPS 1970 SJCC Vol. 36, AFIPS Press, Montvale, N.J., pp. 59–78.
4. PL/C—the Cornell compiler for PL/I. Dept. Comput. Sci., Cornell U., Ithaca, N.Y., Aug. 1971.
5. Zelkowitz, M. Reversible execution as a diagnostic tool. Ph.D. Diss., Dept. Comput. Sci., Cornell U., Jan. 1971.