TOWARD AUTOMATIC DEBUGGING OF COMPILERS*

Hanan Samet
Computer Science Department
University of Maryland
College Park, Maryland 20742

Abstract

Automatic debugging is examined in the context of compiler correctness. A system is described whose goal is to prove the correctness of translations involving heuristically optimized code. The class of errors that can be detected and corrected using such a system is also discussed.

Keywords and phrases: debugging, compilers, error detection, error correction, program verification

INTRODUCTION

In [Samet77a] we describe the use of a compiler testing system [Samet75] in detecting errors in heuristically optimized code as well as the prospects for automatically correcting them. This work is motivated by the realization that often there is no a priori knowledge of how certain computer programs are to be optimized. In such a case, there may be a need to resort to heuristics; thereby necessitating a mechanism for verifying that the various attempts at optimization do indeed function properly.

Compiler testing is a technique of proving that given a compiler (or any translation procedure) and a program to be compiled, the translation has been correctly performed. Compiler testing relies on the existence of an intermediate representation common to both the source and object programs. This representation reflects all of the computations performed on all possible execution paths. Given the existence of such a representation, the testing procedure consists of three steps. First, the high level language program is converted to the intermediate representation via the use of a suitable set of syntactic transformations [Samet77b]. Second, the low level program must be converted to the intermediate representation. This is achieved by use of a process termed symbolic interpretation [Samet76] which interprets procedural descriptions of low level machine operations to build the intermediate representation. Third, a check must be performed of the equivalence of the two representations. This check is in the form of a procedure which applies equivalence preserving transformations to the results of the first two steps in attempting to reduce them to a common representation. This technique has been applied to LISP and PDP-10 assembly language.

ERRORS

Errors in the translated program that are caused by the translation process are categorized into four classes. Errors of the first class are detected by the symbolic interpretation procedure while the remaining three classes are detected during the proof procedure as computations are being matched in the two intermediate representations.

(1) Errors pertaining to the well-formedness of the program include improper calling sequences, illegal stack pointer formats, illegal operations on certain high level data structures, etc.

(2) All of the computations in one of the intermediate representations were found to exist in the other representation, but the reverse is not true. Such an error may arise when certain side effect computations occur in one of the programs and not in the other. Alternatively, this may also arise when certain tests are performed in one program and not in the other.

(3) There are occasions when each of the intermediate representations reflects the performance of the same computations along each execution path, yet, the two representations are not identical. This occurs when the results of the execution paths are different.

(4) The actual proof procedure may reach a point at which it cannot continue. This is the case when a function in the intermediate representation of the low level program can not be matched with a function in the intermediate representation corresponding to the original high level program.

Error correction is a difficult task. In [Samet77a] we show how the following heuristics are used to debug an incorrectly translated complex function (105 instructions). Whenever an error occurs in a function, we determine if the error is caused by the wrong function being applied to a set of arguments or the correct function being applied to the wrong set of arguments. Our approach is first to attempt to correct the function. Next, an attempt is made to correct the arguments. When correcting arguments, we know the accumulators which must contain the arguments and thus we can work backwards to determine where and when the wrong values were computed and loaded into the accumulators. Often the debugging process is aided by the presence of instructions that manipulate data that will no longer be referenced in the program. Such instructions often serve as candidates for removal and replacement by the correct instruction. Errors also occur frequently in testing the wrong sense of a condition. This is especially common with arithmetic relations such as less than and greater than. Such occurrences are signaled by the presence of errors in both subtrees of a condition in close proximity (in terms of the logical flow of the program) to the instruction at which the condition is tested. This can be corrected in the following manner. Reverse the sense of the test. If all of the errors disappear, then the diagnosis is clearly correct. If some of the errors disappear, then the diagnosis is quite likely to be valid. The previous is especially true if at least one error in each subtree disappears after making the change. Note that changing the sense of the test may lead to new errors. However, as long as some of the current errors disappear, the correction is likely to be valid.

The above heuristics were used in a manual process to debug the erroneous program. We believe that many of these errors could be corrected automatically. However, there is a need to continue exercising the proof system with erroneous encodings to determine if any more error correction heuristics can be discovered.

REFERENCES

[Samet75] - Samet, H., "Automatically Proving the Correctness of Translations Involving Optimized Code," Ph.D. Thesis, Stanford Artificial Intelligence Project Memo AIM-259, Computer Science Department, Stanford University, 1975.

[Samet76] - Samet, H., "Compiler Testing Via Symbolic Interpretation," Proceedings of the ACM 29th Annual Conference, 1976, pp. 492-497.

[Samet77a] - Samet, H., "A Study in Automatic Debugging of Compilers," TR-545, Computer Science Department, University of Maryland, College Park, Maryland, 1977.

[Samet77b] Samet, H., "A Normal Form for Compiler Testing," Proceedings of the SIGART-SIGPLAN Symposium on Artificial Intelligence and Programming Languages, 1977.

*This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC 15-73-C-0435. The views expressed are those of the author.