Refining Data Flow Information Using Infeasible Paths*

Rastislav Bodík, Rajiv Gupta, and Mary Lou Soffa
Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA

Abstract. Experimental evidence indicates that large programs exhibit significant amount of branch correlation amenable to compile-time detection. Branch correlation gives rise to infeasible paths, which in turn make data flow information overly conservative. For example, def-use pairs that always span infeasible paths cannot be tested by any program input, preventing 100% def-use testing coverage. We present an algorithm for identifying infeasible program paths and a data flow analysis technique that improves the precision of traditional def-use pair analysis by incorporating the information about infeasible paths into the analysis. Infeasible paths are computed using branch correlation analysis, which can be performed either intra- or inter-procedurally. The efficiency of our technique is achieved through demand-driven formulation of both the infeasible paths detection and the def-use pair analysis. Our experiments indicate that even when a simple form of intraprocedural branch correlation is considered, more than 2% of def-use pairs in the SPEC95 benchmark programs can be found infeasible.

1 Introduction

Static analysis is an integral component of many software engineering tools. Because static analysis is performed before execution, it is necessarily conservative in its assumptions. One commonly made assumption is that every program path is executable. However, some of the paths may be infeasible in that there is no input for which the paths will be taken. Thus, the static analyzers produce imprecise information.

Imprecision in the analysis information results in undesirable consequences in software engineering applications, particularly in testing and debugging. In path testing, paths may be selected for testing which are, in fact, infeasible. In data flow testing, imprecision may lead to the selection of definition-use (def-use) pairs which are impossible to test because they lie on infeasible paths. Considerable effort may be wasted in trying to generate input data, either manually or automatically, that traverses the infeasible paths [9].

Knowledge about infeasible paths can be used to improve the precision of static analyzers because these paths can be excluded from consideration. Although it is impossible to solve the general problem of identifying all infeasible

* Supported in part by Hewlett Packard, the National Science Foundation PYI Award CCR-9157371, and Grant CCR-9402226 to the Univ. of Pittsburgh.
paths, some can be determined by detecting static branch correlation. A conditional branch has static correlation along a path if its outcome can be determined along the path from prior statements or branch outcomes at compile time. For example, along a given path, the direction of a branch may be determined from a constant assignment to the variable that is tested in the conditional, or from the outcome of another branch. Experiments show that from 9 to 40 % of conditionals in large programs exhibit correlation that is detectable at compile time [2]. This implies that a significant number of infeasible program paths can be detected prior to program execution.

Although the infeasible path information can be used to sharpen many tools that are based on data flow analysis, it is particularly useful for software engineering applications, including the following:

- The infeasible path information can be directly used by path testing. In path testing, the algorithm for selecting paths to be tested can avoid paths found infeasible due to branch correlation and thus reduce the effort to generate test cases. Typically, such algorithms do not consider infeasible paths [7, 16].
- In def-use testing, def-use pairs that occur only along infeasible paths can be eliminated from the set of requirements to be covered by test cases. Since 100% test coverage can rarely be achieved on real programs due to presence of infeasible paths, reducing the number of infeasible def-use pairs increases the confidence in regression testing [11] and integration testing [4].
- By avoiding the consideration of infeasible paths during static slicing [10, 14, 17], fewer statements are added to the program slice, thus more precisely identifying the potentially erroneous statements.

In this paper we present a static def-use pair analysis technique that avoids identification of infeasible def-use pairs through detection of branch correlation. The technique consists of two algorithms: (1) the detection of branch correlation and identification of infeasible program subpaths, and (2) the def-use pair analysis that excludes def-use pairs spanning the identified infeasible subpaths. (In the remainder of the paper, the terms infeasible path and infeasible def-use pair refer to paths and pairs, respectively, that are found infeasible by our technique.) Both algorithms are demand-driven, which guarantees good analyzer performance because only nodes that may influence branch correlation or def-use pair computation are visited. Since significantly more correlation can be detected interprocedurally, we have developed both intra- and inter-procedural versions of our analyses.

The algorithm for detection of interprocedural branch correlation was originally developed to support a compiler optimization for the elimination of redundant conditional branches [2]. We extend the correlation detection algorithm in this paper to identify shortest infeasible paths and to label the control flow graph with these paths. Techniques for static branch correlation detection have also been developed by other researchers [8, 15]. While these techniques can detect correlated branches, they do not identify the shape of infeasible paths, a requirement for eliminating infeasible def-use pairs. Furthermore, only correlation between pairs of branches is detected, which is not sufficient for identifying