Modifying Test Suite Composition to Enable Effective Predicate-Level Statistical Debugging

Ross Gore and Paul F. Reynolds

University of Virginia, Department of Computer Science
P.O. Box 400740 Charlottesville, VA 22904 USA
{rjg7v,reynolds}@viriginia.edu

Abstract. In order to effectively deal with increased complexity and production pressures for the development of safety-critical systems, organizations need automated assistance in program analysis and testing. This need is intensified for systems that make heavy use of floating-point computations. Challenges related to the use of floating-point computations exist in the fields of testing, formal verification and debugging. While testing and formal verification provide mechanisms to identify possible failures within safety-critical systems, debugging techniques are employed to automatically isolate the cause of the failure. Recent advances in predicate-level statistical debugging have addressed localizing faults due to floating-point computations. Here, we present a methodology to modify the composition of a test suite to enable predicate-level statistical debuggers to more effectively isolate the causes of failures in safety-critical systems. Our methodology makes test suites significantly more effective for a class of debuggers, including those built to address faults due to floating-point computations.

Keywords: causal model, matching, debugging, safety-critical systems.

1 Introduction

The success of experiments involving safety-critical systems including autonomous robots, Next Generation Air Transportation (NextGen), and fly-by-wire spacecraft depends on the correctness of software [1, 2]. Achieving correctness in these systems is significantly more difficult when floating-point computations are used because the desire to employ efficient floating-point computations increases the likelihood of numerical analysis errors [3].

Floating-point correctness creates challenges within the fields of testing, formal verification and fault localization. Formal verification is difficult because the semantics of floating-point computations may change according to factors beyond source-code level, such as choices made by compilers. Testing is difficult because non-deterministic numerical analysis errors can result in difficult to replicate failures. Fault localization is difficult because the values of variables associated with a fault rarely are exactly equal to one another or a predetermined value.

A. Goodloe and S. Person (Eds.): NFM 2012, LNCS 7226, pp. 70–84, 2012.

Numerous studies have shown that among verification, testing and fault localization, fault localization (debugging) takes up the most time in the development process [4, 5]. Recently, there has been considerable research on using statistical approaches for debugging [6-11]. Statistical debuggers require a test suite, execution profiles, and a labeling of the test executions as either succeeding or failing. The execution profiles reflect coverage of program elements. Program elements refer to individual statements or other inserted predicates. The approaches employ an estimate of the suspiciousness of the program elements. Then developers examine program elements in decreasing order of suspiciousness until the fault is discovered.

Here, we are concerned with predicate-level statistical debuggers. All predicate-level statistical debuggers share a common structure. Each debugger uses a set of conditional propositions, or predicates, which are inserted into a program and tested at particular points. A single predicate can be thought of as partitioning the space of all test cases into two subspaces: those satisfying the predicate and those not. Better predicates create partitions that more closely match where the fault is expressed.

In the canonical predicate-level statistical debugger Cooperative Bug Isolation (CBI), three predicates are inserted and tested for each variable x within a program statement: (x>0), (x=0) and (x<0) [6]. In the statistical debugger Exploratory Software Predictor (ESP), these three predicates are complemented with *elastic predicates*. *Elastic predicates* use profiling to compute the mean, μ_x, and standard deviation, σ_x, of the values of variable x. Then, the CBI predicates are complemented with elastic predicates: ($x > \mu_x + \sigma_x$), ($\mu_x + \sigma_x > x > \mu_x - \sigma_x$) and ($x < \mu_x - \sigma_x$) [8].

Elastic predicate debuggers, such as ESP, are the only fault localization techniques, which are designed to target faults due to floating-point computations. Elastic predicates are effective for such faults because the predicates: (1) expand or contract based on observed variable values and (2) do not employ a rigid notion of equality.

The standard suspiciousness metric for a predicate (elastic or otherwise) is the probability of a program Q failing given that a predicate p is true. This probability, Pr(Q *fails* | p=*true*), indicates if predicate p was true during an execution of Q at least once. Given the execution of a test suite, Pr(Q *fails* | p=*true*) is typically estimated by the sample ratio (f_p / (f_p+s_p)), where f_p is the number of tests for which p is true and the system fails and where s_p is the number of tests for which p is true and the system succeeds (does not fail). However, this estimate and other similarly derived estimates of suspiciousness are susceptible to at least two types of confounding bias: *control-flow dependency bias* and *failure-flow bias*. *Control-flow dependency bias* occurs when the conditions specified by a predicate corresponding to a fault cause other predicates to be evaluated during system failures [7]. *Failure-flow bias* occurs when a triggered fault causes the probability of reaching a subsequent statement where a predicate p is evaluated to be the same as the probability of p being true [6, 9].

In previous work, we introduced a causal model that accounts for these biases, to estimate the suspiciousness of a predicate by considering two groups of executions: those where predicate p is true at least once (the treatment group) and those where predicate p is not true (the control group) [9]. The estimate resulting from this model is more accurate than existing suspiciousness estimates, because it accounts for the possible confounding influences of other predicates on a given predicate p.