Model-Based Diagnosis versus Error Explanation

Heinz Riener*
*Institute of Computer Science
University of Bremen, Germany

Görschwin Fey*†
†Institute of Space Systems
German Aerospace Center, Germany

{hriener, fey}@informatik.uni-bremen.de

***Abstract*—Debugging techniques assist a developer in localizing and correcting faults in a system's description when the behavior of the system does not conform to its specification. Two fault localization techniques are model-based diagnosis and error explanation. In this paper we focus on fault localization for imperative, non-concurrent programs. We compare the two fault localization techniques in a unified setting presenting SAT-based algorithms for both. The algorithms serve as a vantage point for a fair comparison and allow for efficient implementations leveraging state-of-the-art decision procedures. We implement the SAT-based algorithms in a prototype tool utilizing a *Satisfiability Modulo Theories* (SMT) solver and evaluate them on mutants of the ANSI-C program TCAS from the *Software-Artifact Infrastructure Repository* (SIR).**

I. Motivation & Overview

Debugging is one of the most time consuming steps when creating software in general and embedded software in particular. Fault localization techniques for isolating faulty program locations in the source code of a program have been proposed. The techniques divide into two broad categories. The first category [1], [2] is based on *Model-Based Diagnosis* (MBD). MBD searches for components of the program that when replaced correct the program's misbehavior. Thus, fault localization is closely related to program repair, i.e., a component is potentially faulty *if and only if* (iff) replacing the component makes the program correct. The second category is formed by *explanation-based techniques* [3] which "explain" a misbehavior by comparing the differences (and similarities) between faulty and correct execution traces.

We compare an MBD-based and an explanation-based fault localization technique. Both techniques allow for several trade-offs. We adopt the techniques to provide a unified setting and to allow for a fair comparison. In particular, the MBD-based technique is similar to the approach described by Smith et al. [2] but focuses on fault localization in software programs rather than hardware circuits. The explanation-based technique is an adaption of *error explanation* [3].

In a theoretical comparison, we show that depending on the program source either MBD or error explanation is superior, i.e., we construct programs for which only one of the techniques is able to exactly pinpoint the fault. This motivates the empirical evaluation of both techniques on fault localization problems.

We present algorithms to formalize the fault localization problem with respect to both techniques into logic formulae. The inputs are the source code of an imperative, non-concurrent program and a formal specification which is either given by local assertions annotated into the program's source or by a reference implementation of the program. The output is a set of potentially faulty program locations which we call *fault candidates*. The fault candidates are determined using transformation into the *Satisfiability* (SAT) problem.

This work was supported by the German Research Foundation (DFG, grant no. FE 797/6-1).

Our SAT-based algorithms are implemented in a prototype tool which uses *Quantified-Free Bit-Vector Logic* (QF_BV) and solves the logic formulae with the aid of a *Satisfiability Modulo Theories* (SMT) solver.

II. Experimental Results

We compare both fault localization techniques experimentally in a case study using the *Traffic Collision Avoidance System* (TCAS) from SIR [4]. TCAS is an imperative, non-concurrent program which implements a collision avoidance system for aircraft in 135 ANSI-C lines. SIR provides 41 mutants of TCAS. Each mutant corresponds to the correct program with injected faults. The mutant contains a simple mistake with respect to the correct program. These mistakes refer to a single or multiple faulty statements.

For the comparison, we compute fault candidates for each mutant of TCAS with both fault localization techniques. All our experiments were conducted on a PC AMD Phenom™ II X4 Processor which has 4 cores with 3 GHz each and 8 GB RAM. We use the 64-bit version of Boolector 1.4.1 as SMT solver. We asses the quality of the fault candidates similar to Renieris and Reiss [5], i.e., we count their distance from the real faults with respect to a reference implementation of TCAS on the *Program Dependency Graph* (PDG). The distance refers to the length of the cause-effect chain the programmer has to examine in order to locate the real fault.

TCAS consists of 247 program locations. On average, our MBD-based algorithm reports 58 fault candidates per mutant. For each mutant the computation finishes in less than 7 seconds. The average of the distance of a fault candidate to a real fault counted on the PDG ranges from 1 to 7. Our explanation-based algorithm reports on average only 8 fault candidates. However, the maximal time required to finish explanation for a mutant is 79.8 seconds. The average distance of a fault candidate to a real fault counted on the PDG ranges from 3 to 4. Thus, the MBD-based algorithm computes a large set of fault candidates which usually contains one fault candidate close to the real fault. The explanation-based algorithm computes less fault candidates. However, those fault candidates have on average a higher distance to the real fault.

References

[1] R. Reiter, "A theory of diagnosis from first principles," *Artificial Intelligence*, vol. 32, no. 1, pp. 57–95, 1987.

[2] A. Smith, A. Veneris, M. F. Ali, and A.Viglas, "Fault diagnosis and logic debugging using boolean satisfiability," *IEEE Transactions on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[3] A. Groce, S. Chaki, D. Kröning, and O. Strichman, "Error explanation with distance metrics," *International Journal on Software Tools for Technology Transfer*, vol. 8, no. 3, pp. 229–247, 2006.

[4] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering: An International Journal*, vol. 10, no. 4, pp. 405–435, 2005.

[5] M. Renieris and S. P. Reiss, "Fault localization with nearest neighbor queries," in *IEEE International Conference on Automated Software Engineering*, 2003, pp. 30–39.